

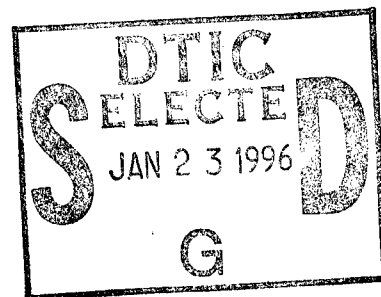
RL-TR-95-118
Final Technical Report
July 1995



PARALLEL SOFTWARE ENGINEERING ASSESSMENT

RCI, Ltd.

Carl Murphy



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19960122 060

DTIC QUALITY INSPECTED 1

**Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-118 has been reviewed and is approved for publication.

APPROVED:



JOSEPH P. CAVANO
Project Engineer

FOR THE COMMANDER:



JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE July 1995		3. REPORT TYPE AND DATES COVERED Mar 93 - Mar 95
4. TITLE AND SUBTITLE PARALLEL SOFTWARE ENGINEERING ASSESSMENT			5. FUNDING NUMBERS C - F30602-93-C-0087 PE - 62702F PR - 5581 TA - 18 WU - PB	
6. AUTHOR(S) Carl Murphy				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) RCI, Ltd. 1301 East 79th Street, Suite 200 Minneapolis MN 55425			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3CB) 525 Brooks Rd Griffiss AFB NY 13441-4505			10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-95-118	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Joseph P. Cavano/C3CB/(315) 330-4063				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Parallel Software Engineering is concerned with the cost-effective development, reuse, and maintenance of high-quality, efficient parallel software. If parallel computers could be used to their full advantage, the capability of C3I systems could be greatly advanced, but developing software for parallel computers intensifies current software problems by reducing portability, increasing software costs, and creating performance uncertainties. This assessment evaluated parallel applications and architectures from the perspective of a C3I system builder and found that parallel computers failed to address all the requirements of Air Force C3I systems. The state-of-the-art in parallel computing concentrates on existing sequential software and parallelizing it to meet performance-driven needs. Better tools are needed to help software engineers provide the generality of real-time constraints. The software development process for parallel computers lacks an integrated set of tools and an architecturally independent programming model. Commercial-off-the-shelf (COTS) components are not available for parallel C3I systems as they are for workstations solutions.				
14. SUBJECT TERMS Parallel software engineering, Parallel processing, Parallel software development			15. NUMBER OF PAGES 104	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

TABLE of CONTENTS

Executive Summary	ES - 1
Scope	ES - 1
Applications	ES - 1
Architectures	ES - 1
Market Force Strategy	ES - 2
Software Engineering Issues	ES - 2
 1 Assessment Overview	 1 - 1
1.1 Topics and Organization	1 - 1
1.1.1 Discussion Focus – Applications and Architectures	1 - 2
1.1.2 Market Forces and Issues	1 - 2
1.1.3 Software Engineering Assessment	1 - 2
1.2 Applications	1 - 2
1.2.1 C ³ I Requirements for Parallel Processing	1 - 2
1.2.2 Parallel Applications	1 - 2
1.2.3 C ³ I Application Requirements	1 - 3
1.2.4 Parallel Applications Classification	1 - 3
1.3 Parallel Architectures	1 - 4
1.3.1 Classification of Parallel Computers	1 - 4
1.3.2 Systems of Computers	1 - 5
1.3.3 Architectural Trends	1 - 5
1.4 Market Forces	1 - 6
1.4.1 Off-the-Shelf	1 - 6
1.4.2 Market Driven Technology Forces	1 - 6
1.5 Software Engineering	1 - 6
1.5.1 Performance Measures	1 - 7
1.5.2 Software Engineering Process	1 - 7
 2 Applications Characterization and Classification	 2 - 1
2.1 Applications	2 - 1
2.1.1 C ³ I Systems	2 - 1
2.2 Assessment of Parallel Applications	2 - 2
2.2.1 Parallel Processing Application Typing and Characterization	2 - 2
2.2.1.1 Response Time Application Typing	2 - 3
2.2.1.2 Fitting Applications to Response Time Types	2 - 6
2.2.1.3 Reducing Response Time by Parallel Processing	2 - 6
2.2.2 Application Characterization	2 - 6
2.2.3 Parallel Application Review	2 - 9
2.2.3.1 Large Scientific Computations	2 - 9
2.2.3.2 Lessons from Scientific Computing	2 - 11
2.2.4 Business Applications	2 - 11
2.2.4.1 Lessons from Business and Commercial Systems	2 - 12
2.2.4.2 Some Successes of Parallel Computers	2 - 12
 3 C ³ I System Focus	 3 - 1
3.1 C ³ I Requirement	3 - 1

Accession For	
NTIS, CRA&I	<input checked="" type="checkbox"/>
DTIC, TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

3.1.1 Effect of Technology on C ³ I Systems	3 - 1
3.2 C ³ I System Limitations	3 - 2
3.3 Critical Point in Industry Progress	3 - 3
3.3.1 C ³ I System Profiles	3 - 3
3.3.2 Recommended Focus on C ³ I Parallel Factors	3 - 7
3.3.3 System Viewpoint	3 - 8
3.3.4 Application Characterization Tools	3 - 8
4 Parallel Architecture Assessment	4 - 1
4.1 Parallel Systems	4 - 1
4.2 Networks of Computers	4 - 1
4.2.1 Systems of Computers - A Necessity in C ³ I Systems	4 - 1
4.2.2 Systems Perspective	4 - 2
4.2.2.1 Clusters and Farms	4 - 2
4.2.2.2 Heterogeneous Systems	4 - 3
4.2.3 Heterogeneous Life Cycle	4 - 4
4.3 Multicomputers & Multiprocessors	4 - 5
4.3.1 Examples of Multicomputers and Multiprocessors	4 - 5
4.3.2 Symmetric Multiprocessors	4 - 7
4.3.3 Multiserver Systems	4 - 7
4.4 Architectural Trends	4 - 8
4.4.1 General High Performance Architectural Trends	4 - 8
4.4.1.1 Microprocessor Trend	4 - 9
4.4.2 Maximum Processor Count Trend	4 - 10
4.4.3 Interconnect Latency and Bandwidth Trend	4 - 11
4.4.4 Obsolescence Trend	4 - 11
4.4.5 Programming Model Support Trend	4 - 12
4.4.6 Capability Trend	4 - 13
4.4.7 Utility Trend	4 - 13
4.4.8 Cost Trend	4 - 13
4.4.9 Real Time Trends	4 - 14
4.5 Architectural Trend Conclusion	4 - 14
5 Market Forces and Issues	5 - 1
5.1 Commercial-off-the-shelf (COTS) Parallel Computers	5 - 1
5.1.1 Strategy for Future Cost Effectiveness	5 - 1
5.1.2 Research Viewpoints	5 - 2
5.1.3 Building Industry Capability	5 - 2
5.1.4 Failure at Generality	5 - 2
5.2 Opportunity and Challenge	5 - 3
5.3 Market Driven Technology Forces	5 - 3
5.3.1 Demand for Commercial Parallel Systems	5 - 5
5.3.2 Acceptance of Parallel Processing in the Marketplace	5 - 5
5.3.3 Technical Computing Marketplace	5 - 7
5.3.4 Commercial Parallel Systems	5 - 7
5.4 Strategy for Embracing the Commercial Successes	5 - 9

6 Status of Software Engineering for Parallel Systems	6 - 1
6.1 The Software Engineering Goal for Parallel Computing	6 - 1
6.2 Process Mismatch	6 - 1
6.2.1 Limited Commercial Applications Successes	6 - 1
6.3 "Sequential" Software Engineering	6 - 2
6.4 Computer-Aided-Software-Engineering (CASE) Review	6 - 3
6.4.1 Comparison of CASE and EDA	6 - 4
6.4.2 Influence on Software Engineering for C ³ I Systems	6 - 5
6.4.3 Software Engineering Project Metrics	6 - 6
6.5 Expressing Parallelism	6 - 6
6.5.1 Parallel Machine Models	6 - 6
6.5.2 Cluster Programming Methods	6 - 9
6.5.3 Heterogeneous System Programming Tools	6 - 10
6.5.4 The Problem of Application Selection	6 - 10
6.5.5 Object Oriented Methods	6 - 11
6.5.6 Graphical Programming	6 - 11
6.5.7 Debugging and Performance Tuning	6 - 12
6.5.8 Performance Tools	6 - 12
6.5.9 Correctness Tools	6 - 13
6.5.10 Visual Tools	6 - 13
6.5.11 Power Languages	6 - 13
6.5.12 Intelligent Interaction	6 - 13
7 Issues in Parallel Software Engineering	7 - 1
7.1 Parallel Software Engineering Issues	7 - 1
7.1.1 Process and Design Metrics	7 - 1
7.1.1.1 Quantitative Performance Assurance	7 - 1
7.1.1.2 Quantitative Program Assurance	7 - 1
7.1.1.3 Development and Life Cycle	7 - 2
7.1.2 Application Metrics	7 - 2
7.1.2.1 Physical Constraints	7 - 2
7.1.3 Performance Delivery & Metrics	7 - 3
7.1.3.1 Evaluation Criteria	7 - 3
7.1.4 Programmability & Portability	7 - 5
7.1.4.1 Programmability	7 - 5
7.1.5 Relation of Needs to Capabilities	7 - 6
7.1.5.1 Suitability Uncertainty	7 - 6
7.1.5.2 Conflicting Measures	7 - 7
7.1.6 Breadth and Market for Applications & Tools	7 - 7
References	8 - 1
A. Appendix A Some successful large scale scientific computing results.	Appendix A - 1
B. Appendix B: Capability Evaluation of Parallel Computers	Appendix B - 1
C. Appendix C: Quantitative Performance and Risk Assessment	Appendix C - 1
D. Appendix D: Connectivity of some parallel architectures	Appendix D - 1
E. Appendix E: Clusters	Appendix E - 1

Parallel Software Engineering Assessment

Executive Summary

Scope

This assessment looks at parallel applications and architectures from the viewpoint of a C3I system builder. Figure ES - 1 shows these issues.

Applications

C3I applications need parallel compute capabilities. Present systems could better fit their physical constraints (size, weight, volume, power, etc.) with effective parallel computing. Parallel processing at the source of data reduces communication. Data fusion into events and information fusion into decisions can be performed faster and more thoroughly. Designers can bring global information to bare on local command decisions and vice versa. Performing multiple simulations and iterations instead of one improves battle plans and scenarios. The reduction in execution time makes each opportunity possible *The contribution of parallel computing is in improving the speed or thoroughness of the result, thus advancing the capability of C3I system applications.*

Our assessment includes a rating of C3I and parallel applications by criteria that are independent of the particular parallel architecture. Together these criteria form a characteristic pattern of execution. By observing successful parallel applications with a similar characteristic pattern, one can identify architectures that are suitable. *The result shows that there are too few successes to meet patterns required by C3I systems. We find that we must make a new focus on parallel architecture capabilities.*

Scientific-research parallel computing is not general enough to provide a complete model for C3I system applications. Client-server data base and on-line transaction processing, a successful example of parallel computing, are limited to applications where the parallel application serves many independent users. *The present parallel successes fail to solve mission critical applications, those in which a few users need an execution that effectively uses the entire capability of the parallel computer.*

Architectures

Gains in throughput processing via parallel processing have, so far, have a significant penalty. Builders find the penalty in the inability to design with assurance that the application can be performed at a predictable speed on the chosen architecture. They experience the penalty when the effort to fit the application into the architecture grows beyond original estimates. The extra effort often compromises

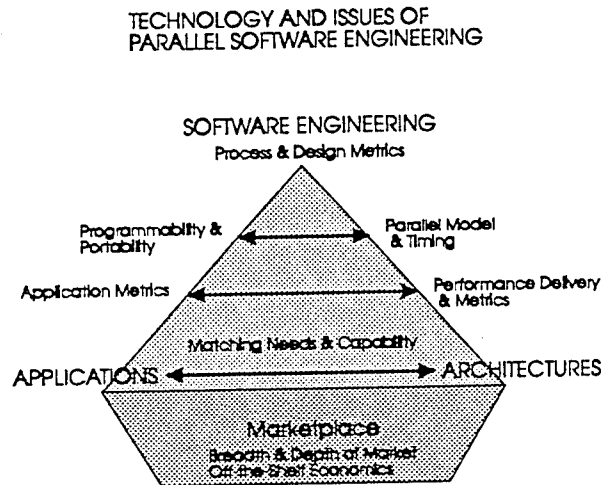


Figure ES - 1. Parallel Software Engineering

the schedule and cost budget. In addition, they have no structured way of designing their system for rapid response computing, real time predictability, system availability, fault tolerance, and security. *We conclude that today's parallel computers fail to deliver the promised performance in complex Air Force C³I systems.* The C³I industry needs a mixture of a robust hardware parallel architecture and software engineering processes that will lead to significant performance leverage, while maintaining programmability, portability and maintainability. The system designer must be able to include physical constraints, time response predictability, fault tolerance and security.

Market Force Strategy

The Commercial-off-the-shelf (COTS) strategy is one in which competition provides advances in capabilities needed for military systems with only marginal investment by the government. COTS components provide an affordable technology for building high performance C³I systems. Often overlooked is that the savings in software costs due to mainstream market acceptance is the most important factor in COTS strategies. *When designers overspecialize parallel architectures to reduce hardware costs, they lose the benefits of COTS provided software.* Operating systems, programming tools and languages, and data base systems are a fall out of commercial successes.

Software Engineering Issues

Software engineering is a discipline that deals with the large, complex, dynamic, mixed hardware of C³I and large scale systems over a long life. Parallel processing technology now exacerbates the development process by reducing portability, increasing software costs and creating performance uncertainties. Parallel computer technology must adapt to attain the performance benefits without the process uncertainty of parallel processing.

The software engineer needs a set of tools that find out the capability and generality of use of parallel computers. Once the engineer knows the adequacy of parallel architectures and the patterns of operation of the C³I system application, the development process can be based on an architecturally independent model. *Separation of hardware and software development is important to the engineering of parallel hardware and parallel software.* Without this capability each effort by hardware vendors and software developers is ad hoc and has little potential for reaching the large markets necessary for commercial advantage.

1 Assessment Overview

This set of essays assesses the status of parallel computing and software engineering for parallel computing. The essays intend to show how parallel computing effects the normal software engineering process for design, development and support of C³I systems. The essays also intend to evaluate the state-of-the-art of parallel computing with respect to supporting C³I system development needs. The essays stress use of parallel computing for reducing the execution time of C³I system components - time response is a critical factor in C³I systems.

- ◆ Application Characterization and Needs
- ◆ Parallel Architecture Capabilities
- ◆ Strategic Use of Market Forces
- ◆ Parallel Software Engineering Issues
 - Process and Design Metrics
 - Application Metrics
 - Performance Delivery & Metrics
 - Programmability & Portability
 - Relation of Needs to Capabilities
 - Breadth and Market for Applications & Tools

TECHNOLOGY AND ISSUES OF
PARALLEL SOFTWARE ENGINEERING

SOFTWARE ENGINEERING
Process & Design Metrics

Programmability & Portability ↔ Parallel Model & Timing

Application Metrics ↔ Performance Delivery & Metrics

Matching Needs & Capability

APPLICATIONS ↔ ARCHITECTURES

Marketplace
Breadth & Depth of Market
Off-the-Shelf Economics

Each corner in the figure is a major topic. Software Engineering issues relate these major topics. The focus is upon those parallel issues that lead to a software engineering process with quantitative

planning, design, and development. The process must allow effective building of C³I applications within architectural capabilities. The marketplace effect is the complicating underside of the relationships - no one can design computer systems without consideration of this dynamic and powerful market.

1.1.1 Discussion Focus – Applications and Architectures

Software engineering practitioners provide the procedures for design, development and long term support within the conflict between applications needs and architecture capabilities. Figure 1-1 shows the Application-Architecture axis. That axis forms the basis of discussion for assessing the state-of-the-art of parallel computing.

1.1.2 Market Forces and Issues

Budgets and risk considerations add constraints in building C³I systems. These constraints prevent a purely theoretical solution. Therefore, a further influencing factor in the discussion is the market for parallel applications and architecture systems. The assessment deals with the present state of applications and available architectures, and not theoretical research computers. Figure 1-1 shows that market force issues are part of the assessment.

1.1.3 Software Engineering Assessment

Software engineering processes provide the process and mechanisms for building large complex systems. Parallel computing leads to different design, programming and performance prediction models from the sequential process. Conventional software engineering processes also target a different model of development. The assessment considers the issues for classifying, evaluating and measuring applications and architectures. The goal is to find out their impact on the software engineering process. Finally, software engineers need to know the capability of architecture types to meet application needs. Figure 1-1 shows the relationships found in the analysis.

1.2 Applications

1.2.1 C³I Requirements for Parallel Processing

Designers and builders of C³I systems generally have a specific mission problem with specific physical constraints. To create a design the mission is typically broken into component subsystems. Each component has a period during which its execution must complete to act in concert with the other components. Parallel computing offers both a reduced response time and improved capabilities because more computation can complete within the system's time constraints.

1.2.2 Parallel Applications

A reduction in the response time for a wide range of application types allows opportunities for improvements in C³I system capabilities.

These essays focus on the technology of parallel processing. This stresses tightly-coupled, self-contained multiple processor computers. C³I subsystems are built from these multiprocessor

components. The assessment also includes distributed computing technology of building multiple computer systems. That technology is needed to construct large, complex C³I systems.

Parallel computing in C³I systems has many sizes and forms. C³I systems use parallel computers for high throughput computing for many independent operations and for rapid response for a single application. Therefore, the essays consider a wide range of parallel multiprocessor types. The C³I system need for information and knowledge processing increases the importance of information systems. Therefore, the assessment includes commercial and business decision making parallel computers along with technical parallel computing.

1.2.3 C³I Application Requirements

Despite a significant investment by the government, the available parallel computers have failed to address all the requirements of Air Force C³I systems.

Command, Control, Communications and Intelligence (C³I) systems must respond to multifaceted threats and scenarios. These systems contain a variety of computing types and subsystems. Parallel processing is necessary to improve both throughput and response time demands. Parallel processing needs improvement in availability, real time response, security, fault recovery, and physical parameters. The system software needs of parallel processing includes operating systems, languages, debugging tools and controls consistent with parallel machines. It needs reliable, portable and reusable software. Present programming processes are too costly and difficult. Air Force C³I application complexity makes it difficult to use the present generation of high performance parallel hardware, system software and applications. A key ingredient in the difficult is the design of predictable rapid response components for complex, unstructured elements of applications.

1.2.4 Parallel Applications Classification

The rating of parallel applications by characteristics shows that few of the available successes are suitable for the characteristics of C³I systems. The industry needs a new focus on parallel computer systems that can adequately support complex and dynamic parallel applications.

C³I systems have some characteristics similar to business and scientific research ones. However, they have a broader set of characteristics than the more narrowly focused applications supported by commercially available parallel computers. This essay proposes a classification scheme. The first typing is by the length and degree of time guarantees for completing an application. The classification extends the one used by Furtney and Taylor to include C³I requirements. The time-of-execution types we propose are the following:

- ◆ Continuous Data Rate (maintaining a constant level of data throughput)
- ◆ Hard Real Time (guaranteed maximum response time)
- ◆ Response Time Goal (graceful operation during occasional failure to meet response time)
- ◆ Interactive Single Job (visual display of results and rapid response to single user)
- ◆ Interactive Multiuser (meets combined delay and throughput benchmark)
- ◆ Capacity Constrained (delays of several minutes are allowed to respond to user)

- ◆ Capability Constrained (results are not needed for immediate use; delays of hours or days allowed; used to improve quality of system operation)

The C³I classification extends the one proposed by Worlton for scientific computing. That classification splits each of four criteria into halves and rates applications on their fit to the resulting sixteen criteria. The four criteria are:

- ◆ Degree of Parallelism (High or Low Concurrency)
- ◆ Parallelism Uniformity (High or Low Uniformity)
- ◆ Grain of Synchronism (High or Low Grain)
- ◆ Communication Distance (Local or Global Distance)

Worlton's criteria is too simple for consideration in rating C³I applications. We propose that the following additions are necessary:

- ◆ Resource Demand Fluctuation (Static or Dynamic Resources)
- ◆ Complexity (Intensive or Extensive).

Typical C³I systems consist of many components, each with a response time goal or hard limit. Predictable response time is critical to our interests. Therefore, our focus is on response time reduction for very general, concurrent applications. These span all the time response types. Unfortunately, scientific processing application characterizations apply to the parallel computers that are specialized for structured scientific applications. These are typically limited to Capability Constrained time response types. A business oriented characterization would be limited to Interactive Multiuser (for on line transaction processing) and Single User (for decision support data base access) time response types.

The C³I application classification extensions for execution time type, resource demand fluctuation, and application complexity are significantly more general than the restricted time response and characteristic classifications and the computers that are build for specialized application needs. This shows that a C³I application characterization must provide very general speed improvements, and cannot be specialized to selected needs. Therefore, the C³I research investment should concentrate on achieving dependable across the board response time improvement and on multiprocessors that can respond rapidly for a wide range of general applications. That part is the missing element in today's parallel software techniques.

1.3 Parallel Architectures

This assessment includes the following subjects:

- ◆ Brief Classification of Parallel Computers
- ◆ Systems of Computers: Parallel Processors, Clusters and Heterogeneous Systems
- ◆ Architectural Trends

1.3.1 Classification of Parallel Computers

Due to their more tightly-coupled structure, multiprocessors have the potential to reduce the execution time of concurrent applications. Our classification stresses communications content and latency that match the parallel system to its potential for a dependable reduction in an application's response time. These are:

- ◆ networks-of-computers - for selected applications with restricted communications and loose time constraints (Capacity and Capability)
- ◆ multicomputers (cost-scalable) - for selected applications with limited communication and time goals over a narrow size, structure, and dynamic range
- ◆ multiprocessors (capability-scalable) - for more general applications with some communications and time limits, capable over a wider range of application structure, dynamic operation and sizes
- ◆ symmetric multiprocessors (count-limited) - for servers in distributed systems with relaxed time limits and for applications with time constraints that can be met with small processor counts

A software engineering classification ignores the structure of the computer and concentrates on the capability to support general applications.

1.3.2 Systems of Computers

There are many competing parallel technologies. Client-server systems, clusters and heterogeneous computing methods provide choices for meeting computing needs. The parallel system software engineer must address the allocation of requirements between parallel processors, clusters, networked computers and heterogeneous systems. These choices form a hierarchy of applications that must match the computer architecture's interconnect capacity. Understanding how a language or tool fits into the hierarchy is critical to evaluation of its value.

Industry understands the software research needs of throughput computing. Applications are On-Line-Transaction-Processing (OLTP), Data-Base-Management-Systems (DBMS), and replicated applications. Clusters and networks of computers are effective for high-throughput processing of independent tasks and replicated processes. Heterogeneous computing mixes network and tightly coupled computing. One form even breaks an application into specialized parts to match specialized processors.

1.3.3 Architectural Trends

Architectural trends are toward a dynamic and multifaceted industry that is on the verge of establishing itself as a major market. Competing forces arise from multiple vendor sources: "scalable" parallel systems, symmetric multiprocessors, commercial on-line-transaction processing, data-base-management-systems, and mainframe vendors. Trends are toward the following:

- ◆ a smaller maximum processor count
- ◆ higher bandwidth capacity interconnects
- ◆ hardware supported programming enhancement mechanisms
- ◆ increased capability for input/output, memory, and disk access
- ◆ increased availability

The trend is also toward improving programmability. Costs per processor node are tending upward as vendors target an engineering production instead of a research laboratory market.

1.4 Market Forces

The discussion of market forces includes the following:

- ◆ Using Commercial Off the Shelf Parallel Processing Resources
- ◆ Market Driven Technology Forces

1.4.1 Off-the-Shelf

Commercial-off-the-shelf (COTS) components are important in high performance C³I systems.

The economy of scale of commercial hardware and software leads to wide spread use of COTS workstations. As a result operating systems, data base systems and high level languages are readily available for scientific and commercial uses of workstations and personal computers. The success of workstations and their ever increasing performance and programmability leads to the same high expectations for parallel processing. Commercial parallel processing industry success is important to the use of parallel processing in C³I systems. C³I system developers need economy-of-scale hardware and software for parallel system, applications, tools, and development processes. Presently parallel processing attracts too few independent software vendors.

1.4.2 Market Driven Technology Forces

There is a constant, cost-driven pressure to move to less costly semiconductor technology. That trend results in reduced demand for large mainframes and supercomputers and an increased one for microprocessor based parallel computers.

Highly competitive markets drive the technology of computing. As a result, technology significantly affects the application of parallel computing. The parallel machine OLTP and DBMS industry shows the success of large processor count parallel computers in multiuser applications. Business information system builders now embrace small-processor-count symmetric multiprocessors for those applications. (Both Compaq and Sun sell over 10,000 units per month as file and query servers.) Workstation, symmetric multiprocessor and scalable parallel vendors now compete in the same market. This convergence shows that parallel computers have market viability. However, few independent software vendors are aggressively porting functional applications to parallel computers. Until they do, the market may remain viable but constrained.

1.5 Software Engineering

The software engineering assessment includes the following:

- ◆ Performance Measures for Parallel Processing
- ◆ Software Engineering for Parallel Systems

1.5.1 Performance Measures

In sequential computing there is a quantitative measurement capability based on instruction timing and the mix of instructions in a typical application. In parallel computers, performance benchmarks and application suites are not adequate.

Workstation designers use application benchmarks and quantitative measures (SPECmarks) to guide development of microprocessors and workstations. Parallel computer designers have no similar quantitative foundation to allow performance estimates for different machines. Parallel computers have no accepted set of commonly accepted instructions upon which to base the measures. Researchers continue to devise nonportable mapping schemes that apply only to a restricted type of applications. Amdahl's law continues to apply in spite of anecdotal results of restricting application types to defeat it. As a result, parallel computers have not escaped the trap of low efficiency and unfulfilled promises of peak speeds.

1.5.2 Software Engineering Process

The software engineering process needs a programming model that is architecturally independent and quantitative. For these reasons, the software engineering process lacks an integrated set of parallel tools.

Software engineering for parallel systems suffers significantly from a too rapid obsolescence of hardware, a lack of common programming models, and the requirement to map to obtain reasonable performance. The general process of software engineering has evolved as a basis for managing the building of long-lived, complex C³I systems. Software engineering processes for conventional sequential computing systems have a built-in factor-of-safety from the progress of technology. The obsolescence of parallel computers, the long procurement cycle, and the lack of an adequate quantitative model for performance assessment of parallel computers negatively affects the software engineering process.

Parallel Software Engineering Assessment

2 Applications Characterization and Classification

2.1 Applications

Software engineering is a discipline that deals with the large, complex, dynamic, mixed hardware of C³I systems over a long useful life. Parallel processing technology now exacerbates the C³I engineering process by reducing portability, increasing costs and creating performance uncertainties. C³I systems need to use parallel processing to meet response time, physical constraints, and new mission functions. Parallel computer technology must adapt to attain the performance benefits without the process pain of parallel processing.

2.1.1 C³I Systems

The goal is to devise a mechanism to characterize C³I system components. If applications fit into types, parallel computers succeed or fail based on how well they execute for a particular type. This means that anecdotal evidence is less subjective. If this can be done without concern for the exact details of the component, then a degree of separation forms between architectures and programming.

Command, Control, Communications and Intelligence (C³I) systems are large systems that must respond to multifaceted threats and scenarios. C³I systems typically deal with complex warfare situations and dynamic scenarios. They respond to multiple forces, to dispersed geographies and unexpected threats across a spectrum of weapon technologies. C³I functions are both highly complex and highly concurrent. They include: transformation of multisource data into information, fusion of information into intelligence, automated and human controlled decision making, simulated scenarios, force and weapon's system responses, communications to higher or lower command levels, etc. As a result C³I System components include a variety of subsystems and interactions:

- ◆ Real time - sensor data conditioning, acceptance, and preprocessing
- ◆ Data Communications
- ◆ Information Base Access
- ◆ Information Fusion & Evaluation
- ◆ Event Detection Decision
- ◆ Scenario Based and Real Time Simulation
- ◆ Action Path Decision
- ◆ Response System Control

C³I systems are heterogeneous multiserver environments. Designers construct them ad hoc, with mixtures of special hardware and commercial hardware to meet specific system needs. Interaction is necessary with humans, networks, sensors and weapons. Burdens of responsiveness, availability, fault tolerance and recovery, security, and reliability, are severe

in C³I systems. Due to their size and complexity software engineering processes are necessary for building these systems and maintaining them over their life cycle.

Performance and programmability on complex, concurrently operating components of a system are important to C³I systems. Building such systems, requires multiple organizations, large programming teams and a hierarchy of skills and knowledge. The systems operate and improve over a long period - their life cycles. These systems contain a mix of computer types ranging from specially developed parallel processors to commonly available workstations. The result is a heterogeneous system. Airborne or fixed locations across global areas are common. Distributed system technology results.

2.2 Assessment of Parallel Applications

2.2.1 Parallel Processing Application Typing and Characterization

The software engineering aspiration is that one can derive a qualitative indication of the suitability of marketplace successes in C³I systems. Therefore, the classification is a way of matching similar execution characteristics among applications in different fields. It is a starting point in separation of parallel architecture design and application programming.

This assessment essay reviews parallel processing applications. C³I system components have a widely varying requirement for execution completion time. Therefore, the discussion provides a C³I application typing by response time requirement. The typing begins with one proposed by Furtney and Taylor and extends it to type that apply to C³I systems. The typing is important because the goal of parallel processing within C³I systems is to achieve reduced response time. That may allow engineers to improve a C³I system, meet physical constraints, or even execute up to performance specification. Another parallel computer use is to improve fault tolerance by physically distributing execution or verifying valid results. The typing is orthogonal to that aspect and use of parallel computing.

Next the characteristics of applications derive from their execution structure. The starting point for that evaluation is one proposed by Worlton for scientific applications. The added parameters of dynamic operation and complexity extend the characterization to C³I systems. The essay reviews the present state of parallel applications and characterizes some successful applications. We relate these applications to C³I application classes.

The result shows that present research computing methods do not apply to the general needs of C³I system components. Research scientists have a significantly higher tolerance level for the extra effort and frustration of mapping their application to a computer. Their successes tend to be well-understood ones in the signal, image and initial data processing stages of C³I systems. The state-of-the-art in parallel computing architecture contributes little to better C³I systems. The C³I systems must be too complex, too dynamic, too real-time constrained and too reliable to make good use of scientific research computers.

2.2.1.1 Response Time Application Typing

Furtney and Taylor, IEEE Spectrum May 1993, identify three types of applications. Their labels go according to the time scale allowed for the computation. They used the following terms:

- ◆ Interactive - human control of display - only momentary waits allowed
- ◆ Capacity or Minutes Frame - uses full capacity of a machine - human would do other tasks while waiting for the results
- ◆ Capability or Hours Frame - problem is significantly larger than full capacity - long time delays, possibly overnight

By extending from a research scientist's point-of-view to a C³I system one, the time response labels expand to include dynamic operation and complexity. C³I systems include many subsystems. This means that the constituent parts often have a different response-time type. Table 2 - 1 shows a proposed C³I oriented typing.

TABLE 2 - 1 Application Types (Extended from Furtney and Taylor)	
Application Types by Response Time	Characteristic
Capability Constrained - Hours Frame (Uses capability of largest parallel machines)	Long turnaround time allowed, could use the maximum capability of parallel computing available
Capacity Constrained - Minutes Frame (Full Capacity of a required for up to an hour)	A delay of several minutes or tens of minutes allowed for the computation (need several iterations per day)
Interactive Multiuser *	multiple users operate on each processor, sharing the processor resource gives a high transaction rate
Single User Interactive*	yet a goal is set for interaction with users Single Job Interactive * fast interaction with a single user is necessary
Response Time *,	Response Time Goal - Graceful response to failure to meet time
Hard Real time *	Guaranteed maximum response time
Continuous Data Rate *	Perform a continuous process at a minimum input data rate; reduce the input rate to a lower output rate; (buffers typically provide continuous rate if time frames are missed)

* Shows an addition to the Furtney and Taylor application typing necessary for C³I.

Table 2 - 2 rates various applications. Parallel computing contributes in the following ways:

- ◆ to improve the performance of an application,
- ◆ to transform it from one time response type into another
- ◆ to radically change the use of the application

A radical change in the use of the application often means moving it from one response-time type to another.

The change could allow novel uses of the application within the C³I system because the application meets a new time scale. For example, if a simulation of a scenario on a workstation lasting several hours executes in a few minutes, then multiple scenarios could be checked out. This could significantly improve the battle plan. Moving weather forecasts from Capability to Capacity Constrained could make them important in tactical battle plans.

Table 2 - 2 Some Parallel Applications by Response Time Type

General Category	Examples	Usual Response Time Type
<u>SCIENTIFIC</u> Large Scale Scientific Research Codes - Grand Challenges	Weather Modeling Climate Prediction	Hours to Day Frame (Capability)
<u>TECHNICAL</u> Engineering Applications	Semiconductor modeling Electronic Design Mechanical Design Chemical and Molecular Modeling Fluid Dynamics Combustion Biomedical Composite Manufacturing Design Materials	Wanted to be Minutes (Capacity) but most often Hours (Capability)
<u>BUSINESS</u> Client Interaction Applications	Customer Interaction OLTP Data Base Management System	Multiuser Interactive Multiuser Interactive Multiuser Interactive
Business Reaction	Knowledge Based and Neural Network Decision Automation Data Base Management System	Single Job Interactive Response Time Goal
Business Decision Support	Strategic Decision Query Data Mining	Hours to Days Time Frames (Capability)
Communications Applications	Voice Response Switches Multimedia	Constant Data Rate (for Voice) Response Time Goal (Telephone) Multiuser Interactive

<u>COMMAND, CONTROL, COMMUNICATIONS & INTELLIGENCE</u>		
Image Processing & Understanding	LIDAR	Constant Data Flow (Processing) Response Time Goal (Understanding)
Command Decision Support		Single Job Interactive / Response Time Goal
Information Base Access	Shared Situation Data Base	Multiuser Interactive
Data Fusion	Target Tracking	Response Time Goal
Information Fusion & Evaluation	External Events combined with Target Events	Single Job Interactive
Event Detection Decision	Event Matching	Automatic - Guaranteed Response Time Manual - Response Time Goal
Action Path Decision	Command Scenario Matching	Response Time Goal
Planning and Scenario Simulation	Battle Plan Preparation	Single Scenario - Hours Frame (Capability) Multiple Scenario - Minutes Frame (Capacity)
Real Time Simulation	Event Projection & Prediction	Response Time Goal
Process Control	Weapon System Control Aim & Fire Missile System	Guaranteed Response Time (Real Time)
Visualization	Battle Situation Display	Interactive
Real time - Data Communications	Distributed Identification Data Delivery	Guaranteed Response Time (Real Time)
Real Time - sensor data conditioning, acceptance, and preprocessing		Guaranteed Response Time

2.2.1.2 Fitting Applications to Response Time Types

Changing the performance of the underlying computer can change a capability type requiring several hours into a capacity one that completes in a few minutes.

A given computation can fit into any of the response time types. For example, decision making using a massively parallel computer might allow multiple iterations of a battle plan instead of a single plan per day. A C³I system design includes a mix of these response time types. The designer characterizes each subsystem by a response time type. Hard real-time response within one subsystem doesn't force all subsystems to be of that type. Often the subsystems are relatively independent. This means that a weapon control subsystem with a real time response requirement is independent of a subsystem that uses simulations to plan a mission planning and scenario operations. Both may be part of the complete C³I system. Their response times can be reduced independently by application of parallel methods. Therefore, they can be used to radically change mission requirements and can drastically improve the system's war fighting capability.

2.2.1.3 Reducing Response Time by Parallel Processing

Our goal is to apply parallel processing methods to reduce component response times within a C³I system into the type required by mission goals and system design.

This often includes fixing the number of processors and accepting constraints on the power, volume and size of the system. The approach is to isolate the applications that make up a C³I system. One rates them by breaking them into response time types and observing the parallel processing opportunity for meeting their response time goals. We can then evaluate the benefits of parallel processing for the subsystems where parallel speed up brings an advantage to the system. Therefore, the C³I goal is to use parallel processing to reduce each subsystem into its mission fulfilling response time type. We then characterize the computation to observe the suitability of a parallel processor.

2.2.2 Application Characterization

Worlton and Associates have published a classification of scientific computations based on scale of parallelism, uniformity of communication, distance of communication, and granularity of synchronization. Figure 2-1 gives the sixteen-way division made if each of these classifiers is split into two parts and graphed orthogonally.

- Parallelism uniformity represents the fluctuation in the concurrency during the computation. In some systems, this is a measure of the differences in computational length of the tasks. The single program, multiple data programming style represents this highly uniform parallelism. Programmers map data to each processor where copy of the code is present to provide the computation.
- Grain of synchronization represents the ratio of parallelism operations to ordinary instructions. Note that there are many definitions of Grain but the bimodal evaluation allows one to ignore some detail of the definition.

According to this scheme the ideal parallel application has high uniformity, high concurrency, local communication, and coarse grain. It is a W16 type. The most difficult to benefit from parallel computation is a W6 type, with low concurrency, low uniformity, global communications and fine grain synchronization. Note that the Worlton scheme is for evaluating scientific applications and for comparing parallel computers to vector computers.

C³I system builders have a different set of constraints. For example, the first steps of image processing are a W9 category. Later steps, which operate over the entire image, would be a W10 type because of the change to global communications.

The Worlton evaluation scheme does not support a dynamic switch between types of processing. Adding a dynamic resource demanding application creates a "dynamic resource demand" as a criterion.

In addition, Worlton's typing is directed toward well-structured problems. I call those applications *intensive*, a term taken from game theory for cases where it is possible to analyze all moves. I call applications that are too complex for complete analysis *extensive* ones. Operating system kernels are an example of an extensive application. Therefore, a sixth criterion is added, complexity, to distinguish intensive and extensive complexity types.

Note that input/output capacity and access from the parallel nodes is not included in the criteria list. C³I systems have a balance requirement for I/O defined by the application and systems design. Consideration of I/O capability is necessary for evaluation of a parallel computer within that system. Table 2 - 3 gives the criterion and their impact on system difficulty.

Table 2 - 3 Expanded Classification Scheme for Applications

Criterion	Difficult Aspect (Requires better capability by architecture)	Facile Aspect
Degree of Parallelism (Concurrency)	Low Concurrency	High Concurrency
Uniformity of Parallelism	Low Uniformity	High Uniformity
Grain of Synchrony	Fine Grain	Coarse Grain
Communication Distance (Locality)	Global Distance	Local Distance
Dynamic Resource Demand	Resources Dynamic	Resources Static
Complexity	Extensive (unstructured)	Intensive (well structured)

2.2.3 Parallel Application Review

2.2.3.1 Large Scientific Computations

The MPP ("cost-scalable") multicomputers cannot yet be considered successful in production scientific and technical computation. In addition, their ineffectiveness in smaller sizes, unless there is significant mapping effort, has left them out of the engineering technical computing market.

Scientists use computing as an essential tool. Computer simulations create the opportunity to explore ideas and design practical experiments leading to a more rapid and economical advance in science. Scientific grand challenges do require massive performance. Some suggest parallel computers as the solution, while they see vector-parallel computers as about to saturate in performance gains. The massively parallel processors (MPP) computers resulted, but have failed to work well on the more needed and economically justified smaller applications used by many engineers and scientists. (The assessment essay on parallel architectures defines MPP machines as "cost-scalable multicomputers.")

A large fraction of parallel processing research concentrates on scientific computations. This perspective comes from the role of vector supercomputers at National Laboratories, NASA, and the NSF Centers. In that sense, parallel processing would be successful when it shows displacement of the vector supercomputer in the day-to-day computational operations at those centers.

These computations are large Fortran codes, often developed and run by the same scientists who need their results. This creates a role model for parallel development of individuals creating and using their own creations for scientific studies. Many successes reported are of this type. Appendix A -- Table A identifies some of these successful scientific applications. The tool builders target their products, e.g., High Performance Fortran, to those scientists and their computations. Table 2 - 4 gives results of a NASA scientific kernel for four parallel computers. ["Future Directions in Supercomputing," D. Robb, Cray Research, Supercomputing '93.]

TABLE 2 - 4. NAS Benchmarks (equivalent Cray Research C-90 processors)
(Sixty Four Processors for each parallel system)

	EP	LU	BT	SP	IS	FT	CG
T3D	0.27	0.55	0.48	0.50	0.52	0.49	0.62
SP-1	0.42	0.55	0.43	0.45	0.27	N.A.	0.24
CM-5	0.23	0.2	0.37	0.32	0.12	0.23	0.3
Paragon	0.075	0.11	0.27	0.18	0.07	0.14	0.22

Since this table was published some vendors have significantly changed their systems, claiming that these benchmarks are not representative of their machines. The table states a rating in terms of the fraction of the parallel computer's performance compared to that obtained by a single processor Cray Research C-90 on the same benchmark. Note the dilemma of the multicomputer machines represented by the CM-5 and the Paragon. The EP benchmark, believed to be embarrassingly parallel by the benchmark designer, was not done well by the Paragon. This points out that systems can be deceptive in delivered performance for applications even when their grain of synchronization matches the multicomputer.

The conclusion is that the cost-scalable multicomputer machines, claimed to do well on communications scarce (computational expandable) codes, do poorly on some of them. They also often execute poorly on smaller problems that contain communication content that grows with problem size.

In addition, many common applications contain a significant fraction of code that is not like the core parallel codes. Evaluators give NASTRAN as the example, only one-half of its core matches the parallel model provided by the cost-scalable multicomputer machines. The other half is too complex and dynamic for a single-program, multiple-data computation system.

For these reasons, the cost-scalable multicomputer machines remain in research departments, while vector processors continue to provide the production execution for large Fortran applications. Cost-scalable multicomputer machines do not provide production work for a broad range of scientists even in the most innovative of environments. Since these machines have not integrated well with the normal scientific research processes, they cannot yet be considered successful in the large scale scientific computation. In addition, their ineffectiveness in smaller sizes, unless one makes significant mapping effort, has left them out of the engineering technical computing market. They don't fit into the engineering process either. From the viewpoint of a C³I system designer the chosen applications reported for these machines have little relationship to the needs of C³I systems. However, the next section shows that some methods are useful in the scientific computing arena. The disappointing acceptance of cost-scalable multicomputer machines does not mean that the methods and processes are not valid for some specific C³I needs.

2.2.3.2 Lessons from Scientific Computing

Baskett and Hennessy [Baskett93] describe some parallel techniques used in scientific applications. They give the following problem types and the techniques for obtaining high performance:

- ◆ multipole - treat clusters of particles
- ◆ direct matrix - do blocking and tiling
- ◆ iterative - combine grids from discrete grids (multigrid)
- ◆ spectral - tile

Tiling means grouping of clusters of frequently communicating tasks to reduce communication between processors. Some degree of some "tiling" is necessary to use a microprocessor well. If a code is such fine grain that it destroys locality-of-reference and breaks pipelines the microprocessor will operate ineffectively. However, tiling, beyond that necessary to keep caches warm and pipelines operating, reduces the potential speedup in a parallel application by making the amount of parallelism smaller. For C³I applications, with fixed size, this means that this technique limits on the number of processors that can be effective on the problem.

2.2.4 Business Applications

Multiuser applications for on line transaction processing and data base accesses are established applications in business systems. The "client/server" mode of processing has allowed a server market estimated to be over \$4 Billion in 1994. Together Sun Microsystems and Compaq sell over 20,000 of these servers per month. Almost all these servers are small processor counts (two to four). They use shared memory for symmetric multiprocessing. The typical applications are multiuser access to data bases. Data-base-management-systems for these computers have been the subject of porting at the "commodity" level, e.g., a Microsoft /Sybase consortium for SQL server software.

Some systems have large processor counts and are local memory based. For example, AT&T provides both types and supports both Oracle and Sybase DBMS systems across the product line. Special programs (e.g., AT&T's Navigator products) allow execution of large complex queries in parallel. A breakdown of applications can be guessed from AT&T's portions given in Table 2 - 5.

Table 2 - 5 Diverse Areas of Business Success in Large Parallel Computers	
AT&T Global Information Systems Application Areas	Portion of Revenues [Smaby]
Telecommunications	36%
Retail/consumer	18%
Banking/Financial	12%
OEM	7%
Government	6%
Airline	4%
Insurance/Health Care	4%
Other	13%

2.2.4.1 Lessons from Business and Commercial Systems

Corporate purchases of small parallel servers for transaction processing and data base accesses show the quick acceptance occurs with application success. Both hardware and software technology in that market are moving toward commodity pricing and volume production. The methods rapidly become understood and applied.

The client server approach is successful primarily in multiuser interaction applications. Symmetric multiprocessor servers have displaced business applications that were formerly on large mainframes. The reported cost savings over mainframes varies from a factor of three to twenty. Large-scale, decision-support applications have a more limited success. There are several examples that provide a good indication that there is excellent potential for doing large applications of the capacity type. One lesson is that there is a difficult integration process with heritage systems. Working with legacy systems built on proprietary data management and operating systems inhibits use on large-scale, decision-support applications.

An application expected to require large scale parallel computers are the switches necessary for multimedia communication switches. Large parallel computer switches will follow the asynchronous transfer mode (ATM) standard specification. Telephone PBX exchanges have been special purpose parallel computers but more general-purpose parallel computers provide part of this new demand.

2.2.4.2 Some Successes of Parallel Computers

Most of the successes and role models created by parallel researchers are class W16s. Since many applications at the National Laboratories are of that type there has been a rush to serve that special market. The W16 applications have high concurrency, coarse grain, local communications, and high

uniformity of parallelism. (See Figure 2 - 1 for the rating definitions.) The W16 additional criteria are steady and intensive. I/O requirements are typically low, because they for Capability Constrained time response problems. (See Table 1 on page 1 for the definitions of F&T typing.) The response-time type is that of Capability when the application is so large that it can take a day if necessary. The characterization of C³I applications placed no computational components in the W16 category. (Note that these ratings are subjective and worth additional debate.) However, they do show a trend or general condition. The research sector has been picking the easy applications, or the available computers do well only on W16 type applications and few others. Usefulness for the computers built to specialize in grand challenge applications is limited. This is due to their specialization for the W16 application type. Market needs for such a specialized machine are too limited to support a parallel industry.

The success of multiuser transaction and data base machines in the business community is in the W11 category, having global communications, high concurrency, and high uniformity. The grain is fine because the model used is a data-base-manager task communicating with multiple user transaction tasks. Communications between the two are high. An analysis of the interconnect structure of large commercial data base machines shows a highly capable interconnect structure. The result is that these communications execute well. Symmetric multiprocessor machines maintain balance by keeping the processor count low. A broader acceptance of symmetric multiprocessors is taking place because of this successful application. Problems with large, complex queries will drive these processors to better performance on a wider range of application types.

Communication switches are another successful example, although most of those presently in use are special purpose PBXs and not general-purpose parallel machines. They typically do telephonic, or multimedia switching. This is a category W11. The criteria fit a high concurrency and coarse grain (the parallel commands are only used to set up a link). The uniformity is high because packet sizes are constant. Its communications are global, since any output port can connect to any input port.

Data collection or signal processing is another example of special purpose parallel machine success. This example falls into category W16. Users and integrators assemble these machines from single board, multiple signal processor computers. One characteristic of these designs is that there is little storage. The system brings in data, converts it and combines it into the necessary signals. Typically, it spectrally processes the signals and sends the results to other programs that do conversion into information that allows decision making code to analyze it. Special purpose processors, not microprocessors for workstations, are dominant in this sector. Multiple TI TMS320C240 processors on a VME bus are typical. Parallel signal processors of this type are found in many C³I systems. A variety of real time operating systems and libraries are available. An interesting observation is that these systems are highly competitive and users can put together large processor count systems for significantly less cost than the available MPP models. For example, board with four i860 processors costs significantly less than a single node of the Intel Paragon. These boards use VME bus for power and system control but have proprietary interconnect structures to build high speed communication paths between processors. Mercury, CSPI and Sky are example manufacturers. These computers are also used in process control systems. Other hardware successes are attached special purpose parallel processors. The MassPar is a typical example attaching to DEC mainframes, minicomputers and workstations as hosts.

Parallel Software Engineering Assessment

3 C³I System Focus

3.1 C³I Requirement

Parallel systems do not yet have the capabilities to meet C³I system needs for rapid response computing, real time predictability, availability, fault tolerance, and security

C³I systems are complex and dynamic. Parallel system research and vendors presently focus on replicated and static applications that fit only a part of the C³I component spectrum. However, we hope to influence the vendors to build systems that meet these needs by focusing on rapid response instead of multiuser-throughput-only hardware and software capability. By bringing that focus to bear for parallel systems with adequate capability, we can make the software engineering process for parallel computers feasible and valuable. Independent software vendors will join in the movement if parallel computers are suitable for the entire spectrum of C³I system needs.

3.1.1 Effect of Technology on C³I Systems

The success of user interface software creates a higher expectation that leads to demands for increased performance and new operational actions and responses. Parallel computers are a solution to many increased performance and capacity demands.

Ever increasing demands for higher performance is a common characteristic of large systems, both military and commercial. New missions and threats require continued performance increases and functional improvements within existing physical constraints. Excellent progress in workstation performance and human interfaces and standards (e.g., Common Operating System Environment or COSE) has led to higher expectations from the user. Now one can create complex requests and demand increased system capacity without realization of the demand's scope.

We need a mixture of a robust hardware parallel architecture and software engineering processes that will lead to significant performance leverage, while maintaining programmability, portability and maintainability.

The potential leverage gained through parallel computing is significant. However, in C³I the multiple processes interact in complex ways. They may synchronize or communicate frequently with each other and have little recognizable structure. However, C³I systems are also highly concurrent. The potential amount of parallel processing done within a given

period is very high, in spite of its unpredictable nature. The potential leverage from parallel processing is present. C³I systems gain an advantage only if the parallel computers are capable and software is available for processing dynamic and complex parallelism.

3.2 C³I System Limitations

Gains in throughput processing made via parallel processing have, so far, suffered a significant penalty. One dimension of this penalty is that the expected availability, portability and programmability are very difficult to achieve. A second dimension is that each generation of parallel hardware is significantly different from its predecessors - leading to reduced portability. A third dimension is that today's parallel computers fail to deliver the promised performance in complex Air Force C³I systems.

The government makes significant investments toward the development of hardware systems, software applications and software tools for building and programming high performance parallel computers. So far these investments fail to address all the requirements of Air Force C³I systems. Specifically they lack availability, real time response, security, fault recovery, reliable software, portability and reusability, etc. In addition, programming processes remain costly, difficult and nonportable among different parallel computers. Users often find that high level programming only works for selected applications that consist of replicated copies and multiple independent tasks. They find reasonable performance efficiencies difficult to attain. System builders usually find a mismatch between the Air Force's dynamic and unstructured C³I applications and the capabilities of today's high performance parallel systems. Commercial system builders find that their distributed servers are good matches only for multiuser "throughput" processing, where multiple transactions and multiple users run simultaneously. Their success has not yet been consistently replicated in complex business decision support systems with applications more closely attuned to C³I systems.

Potentially, parallel processing offers significant operational speed and cost advantages for building C³I systems. However, the present limitations of commercially available parallel computers restrict their use to static and well structured C³I system components. These limitations also lead to expensive development costs and inefficient use of processing capability. In addition, the budgets for C³I systems require more commercial-off-the-shelf and high level coding content than used in the past. Faced with these burdens the builders of C³I systems need both more capable parallel computers and parallel software engineering processes and tools. The combination of parallel systems and software engineering processes and tools must yield reasonable development and life cycle costs and deliver expected performance levels.

3.3 Critical Point in Industry Progress

The Air Force opportunity is to influence this paradigm shift so that the added software tools, applications, and techniques increase the effectiveness and productivity of building C³I systems.

The multiprocessor industry is reaching a critical nexus in its development. Expectations are high that commercial and industrial uses will shift from Emitter Coupled Logic (ECL) supercomputers and mainframes to multiprocessors. The latest commercial vendors have brought out parallel designs that have lower latency and higher bandwidth capacity. Those improvements may allow more dynamic use of the parallel capability. These investments have included higher emphasis on availability, programmability and portability. As a result, advances in technology are creating an opportunity for the C³I system industry to influence these market forces. Some expect that a widespread parallel computing market will allow C³I systems to contain a higher fraction of commercially standard software and network interface components. The Air Force needs to influence vendors to provide the real time, fault recovery, security and other requirements inherent in C³I systems. Vendors (hardware, operating system, language and independent tool and application developers) would consider, and likely include, these C³I factors if well identified and publicized.

A successful strategy for software engineering would be to contribute to processes and tools for development of commercially viable process control, information and engineering systems. These tools should be effective and portable across all the marketplace winners. If there is a need for support of specialized processors from outside the commercial marketplace, then those costs would remain programmatic. Extensions of the commercially viable model to include the added complexity of C³I systems (e.g., physical package constraints) are often necessary. Such extensions would be based on a widely used tool set than on a purely military C³I tool set.

3.3.1 C³I System Profiles

The goals of C³I systems are the following:

- ◆ Reduced Planning and Execution Cycle - Move Capability to Capacity - more iterations per day
- ◆ Decentralize Execution Control - Automatic and Human Local Situation Response
- ◆ Improve Joint Interoperability/Joint/Common Evaluations - Global Views for Decision Making
- ◆ Distributed Command & Control - Communications of Plans, Situation Evaluations, and Actions

The computing technologies necessary for building Command and Control systems are:

- ◆ decision support systems
- ◆ systems engineering and integration
- ◆ distributed computing environments
- ◆ distributed data base management
- ◆ human computer interaction
- ◆ communications networks
- ◆ intelligence exploitation
- ◆ surveillance

The range of processing in C³I systems is large. An Air Campaign Plan guides the operations. The plan includes air battle planning, projected intelligence, battle plan execution, and enemy situation correlation. Those plans require simulations, historic data bases, and integration communications across the command structure. At each level, the users monitor, replan, direct and act upon the battle situation. Some plan elements change interactively, others hourly, others daily.

During execution, some components of the C³I system operate automatically and respond to threats in real time. Others operate on the time frame of human interaction. They display information and interact with humans for decision processing. Surveillance systems collect and process data collected by many mechanisms across a wide spectrum. Multiple types of radar, radio interceptions, identification signals, optical detection, and human visual information are necessary. These signals are processed and integrated with intelligence information to provide a decision information base.

Some components communicate with other C³I systems to obtain a global situation overview to guide subsequent battle decision making. Some actions follow battle plans made on a daily cycle. Often planners must adapt these plans and decision guidance on a short cycle. Ideally, a planner has multiple simulations of the proposed scenarios resulting from a plan. Data bases of assets and personnel are updated and used for planning. They are the basis for making resource requests and reports to higher level commands. Table 3 - 1 gives some processing types that match C³I system computations.

Table 3 - 1 C ³ I Activity and Computational Components		
C ³ I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Signal Processing & Data collection - filtering and preparation	spectral, direct matrix, signal feature extraction, communication	RTT: Continuous Data Rate Concurrency High; Uniformity High; Grain Fine; Distance Local ; Resource Static(high I/O) Application Intensive

C³I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Surveillance, target identification, friendly identification	symbolic feature extraction and interpretation, feature generation - hypothesis learning, feature extraction signal recognition, false signal removal, identification & false alarm detection	RTT: Response Time Goal Concurrency Low; Uniformity Low; Grain Fine; Distance Global; Resource Dynamic Application Extensive
Air Battle Planing: Weapon resource allocation, air battle simulations, scenarios, projection of events, battle planning & history	planning and scenario evaluation, AI, data base, system decision making improvement (AI)	RTT: Capacity Constrained Concurrency High; Uniformity Low; Grain Fine; Distance Global; Resource Static; Application Extensive;
Battle Execution: command response (rapidly varying demands on system), automatic response execution	real time, dynamic capability and fault tolerance system fault correction and recovery (system coordination with other components)	RTT - Hard Real Time Concurrency High; Uniformity Low; Grain Fine; Distance Global; Resource Dynamic; Application Extensive;
Human interaction	decision control, displays, alarm information presentation, human interaction (GUIs, transaction processing)	RTT Interactive Multiuser Concurrency Low; Uniformity Low; Grain Coarse; Distance Local; Resource Static; Application Extensive;
Information Display	graphics, rendering virtual reality (scene projection & computation) sound and voice	RTT - Response Time Goal Concurrency High; Uniformity Low; Grain Coarse; Distance Local; Resource Static; Application Intensive;

C³I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Battle actions	potential event recognition (multidimensional correlation, blackboard), event determination (numerical, multipole, symbolic), decision surface computation (neural networks, AI), multiple event association and data fusion (fuzzy logic)	RTT: Response Time Goal Concurrency High; Uniformity Low; Grain Fine; Distance Global; Resource Dynamic; Application Extensive; (high I/O)
Battle Commands	weapon system command, human overview, command overview	RTT: Interactive Single Job Concurrency Low; Uniformity Low; Grain Coarse; Distance Local; Resource Dynamic; Application Extensive;
Battle replanning: battle resource limits, constraints and projections	linear algebra, resource modeling, data base access	RTT: Capacity Constrained Concurrency Low; Uniformity Low; Grain Coarse; Distance Global; Resource Static(high I/O) Application Extensive
Situation Evaluation, position and geometry correction	filters, transforms, spectral, real time decision processing, data base	RTT: Interactive Single Job Concurrency High; Uniformity Low; Grain Coarse; Distance Global; Resource Static; Application Extensive
Communications, external information exchange, joint operations and multiple users	communication switches, decision making, transaction processing, data base, external oversight, signals via IFFN system	RTT: Response Time Concurrency High; Uniformity Low; Grain Coarse; Distance Local; Resource Dynamic Application-Extensive (high I/O)

C ³ I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Development	System Simulations, Testbeds, Emulators	RTT: Capacity Constrained Concurrency Low; Uniformity Low; Grain Coarse; Distance Global; Resource Static Application Extensive
Life cycle	large development teams, reusable codes, object oriented, Future costs	RTT: Interactive Multiuser Concurrency Low; Uniformity Low; Grain Coarse; Distance Global; Resource Static(high I/O) Application Extensive
Constraints	physical size, power consumption, response time, radiation, fault tolerance, fault recovery, ...	RTT: Real Time + Fault Tolerant + Physical Constraints Concurrency High; Uniformity Low or High; Grain Fine; Distance Local or Global; Resource Static or Dynamic (high I/O) Application Extensive

3.3.2 Recommended Focus on C³I Parallel Factors

The C³I parallel system designer needs tools that adequately characterize parallel C³I applications, C³I based benchmarks that adequately test the architectures now available and suitable quantitative measures of applications that guide hardware designers.

Present successes are usually machine dependent. They are typically coarse grain, uniform parallelism, and local communication. They operate steadily, most often in a single program, multiple data mode. Many types of applications necessary for building C³I systems are missing from the list of successes. The centralization of parallel processing research has led to over-concentration in a few applications instead of supporting many. As a result, there is a workstation viewpoint instead of a systems view. There is a lack of generality in the available hardware. There is a lack of portable applications due to mapping for chosen machines instead of insisting on

general solutions. In depth understanding of the long range use of applications is lacking. There is too much attention to the latest "buzz word" instead of investigating underlying principles.

3.3.3 System Viewpoint

Building C³I systems requires a system viewpoint. C³I builders need a method of typing and characterizing their component computations. The methods used in this essay are too subjective and coarse. By typing and characterizing the computation required in each subsystem, we will make better decisions about applying parallel computers to C³I applications. This is not an advocacy of "heterogenous" computing. The state of the art of parallel hardware and software is too primitive to accomplish heterogenous computing effectively.

3.3.4 Application Characterization Tools

There are no tools that adequately characterize parallel C³I applications. Such tools would allow us to set better research directions. There are no adequate parallel C³I benchmarks or measurement systems that adequately test the architectures now available. (Rome Laboratory has one project in process to define a C³I parallel benchmark.) Most of the present benchmarks come from the National Laboratory scientific research computing model instead of from applications representative of C³I systems.

There are no suitable quantitative measures of applications that guide hardware designers. With the right benchmarks and tools they can create parallel computers for the complex and dynamic execution necessary for large scale C³I systems. We need quantitative measures of parallel machines and applications similar to the RISC efforts pioneered by Hennessey and Patterson. These measures must come from a standard parallel instruction set that encompasses both message-passing and shared-memory architectures.

Only by creating a quantitative approach to C³I system building can architecture development and application programming become an engineering activity. A quantitative approach would allow inadequacies of parallel machines to be identified and corrected and would allow the software engineering community to build its processes upon a firm base.

Parallel Software Engineering Assessment

4 Parallel Architecture Assessment

4.1 Parallel Systems

There are various types of parallel systems, each with its own advantages and disadvantages in programmability, cost, and performance. C³I systems include parallel systems and components of many organizations and types. In addition, the software engineering process might be different for each parallel system type. A parallel system naming that is consistent with the software engineer's needs is appropriate. The use of these names is to ensure that the context of a discussion is understood by all involved. Too often discussions switch from one type of parallel system to another without considering the significant shift in underlying capacity and the wide variation in techniques that might apply to one type and not another. Without a parallel system naming convention there is little chance that the capability can be matched to the time response requirement of an application.

The following parallel system types derive from the software designer's need to consider the underlying interconnect structure speed and bandwidth capacity:

- ◆ networks of computers (linked standard workstations and server computers)
- ◆ multicomputers with constant bandwidth capacity added per node computer (cost-scalable)
- ◆ multiprocessor computers with constant bandwidth capacity between any two processors for any processor count (capacity-scalable)
- ◆ symmetric multiprocessors

4.2 Networks of Computers

Networks of computers are multiple workstations and servers in a loosely coupled local area or wide area network. The protocols used for information transfer between nodes are based on telephonic requirements. Messages go outside one system and into another. Those protocols are efficient for large-size message passing. A dedicated network of homogenous multiple microcomputer nodes is a Cluster or Farm. Often servers of various types are added for enhanced performance. That results in a client server network.

4.2.1 Systems of Computers - A Necessity in C³I Systems

Clusters, multiserver and heterogeneous computing are all components used in building C³I systems and subsystems. All of them require significant systems design and analysis for use as C³I subsystems. Their technology and software engineering techniques are necessary for building C³I systems. The interaction of parallel processors with systems of computers is a critical factor in software engineering of parallel computers. System designers get little objective and quantitative information about performance and responsiveness of a parallel subsystem. Engineers need such information to put parallel computers into either C³I systems or systems of computers.

When over specialized, C³I systems face significant cost associated with development and life cycle maintenance. The desire to use commercial parallel systems comes from the need to avoid this high life cycle cost. Complex C³I systems cannot use currently available commercial parallel systems because their responsiveness and performance on complex and dynamic applications are unpredictable.

Performance and functionality demands are rapidly growing. The reasons are: competition for more responsive reaction, requirements for shorter product development cycles, needs for "whole" process solutions. Parallel systems operating with client workstations reaching networks with high performance servers are a potential solution to meeting high performance needs cost effectively. Providing functionality along with performance on parallel systems is the primary issue of the parallel software engineering process. Client-Server operational modes are typically used to separate the user and the high performance computational engines.

4.2.2 Systems Perspective

Networks of computers, such as clusters, multiserver networks and heterogeneous computer networks are alternatives to parallel multicomputers and multiprocessors. The common network of computers in a C³I system is a heterogeneous one where various types occur. Multiprocessors are often servers in heterogeneous networks, providing data, graphical display and computational capabilities. For highly structured applications, a homogenous cluster is another alternative to a parallel computer. Our purpose is to clearly separate the issues of parallel computing to focus the discussion. Clusters, networks of computers and heterogeneous computing networks are valid C³I ideas, but this assessment focuses on closely coupled parallel computers because of the need to control the response time of the application being performed on the parallel computer.

4.2.2.1 Clusters and Farms

Clusters of workstations are standard workstation processors that connect to each other via a network, not an interconnect backplane. Two forms are available: those built from dedicated, rack-mounted workstation processors (without a display unit for each processor) and those that are networks of desktop workstations. We will refer to the first type as clusters and the second as "farms." Clusters and Farms operate effectively on replicated copies of applications executing the same code on different parameter sets. Clusters also can be dedicated systems that operate as throughput engines supporting multiple engineers, each running a separate copy of an application. Many engineering problems require repeated computation over different parameter sets. System administrators can set up farms to run overnight, also for replicated copies of applications applied to different parameter sets. The Farm approach has appeal to overnight use of "idle" workstations, thus optimizing an organization's use of compute resources.

Clusters provide a choice for many organizations with the need to share an application among users. When a replicated multiparameter application is the only requirement, the cluster also serves well. However, if the user has a large application that does not fit the throughput or replicated application type for which these machine are useful, the cluster is

less appealing. Here the programming effort is expensive and the users must wait significantly longer for turnaround than if a multiprocessor or supercomputer were available. The multiprocessor server is the solution to the performance shortfall of clusters.

4.2.2.2 Heterogeneous Systems

"Heterogeneous" parallel systems are network systems with mixed server types. C³I systems are classical versions of the heterogeneous system. The software engineering processes for network systems deal with complexity of development, acceptance, and long term maintenance. The industry needs development and operational tools that deal with the management of applications within a heterogeneous environment of workstations and conventional servers. Pipelines built of individual applications are a typical solution. Parallel servers add an additional level of complexity because they often have low system input/output capacity. Performance falls off rapidly when a part of an application mismatches a machine's ideal form of the computation. Mapping to match the computer is necessary. Due to the "mapping" portability is very low and performance is brittle. (Brittle means that small changes in the application make significant differences in the time taken for delivery of the result.) Programmers find that performance can fall significantly with even small changes to codes or when a code is ported to another parallel computer.

Researchers have concentrated on how to break up applications into components that match different parallel machines to overcome machine mismatches. The idea is to pass the results of components across the network so that each executes on an ideal machine for its structure. By avoiding any mismatch the whole computation runs faster. The approach assumes that there is an ideal match for each application component in the system and that the programmer or compiler can identify it. However, the obsolescence rate of parallel computers is so great that mapping of application components to different servers is often required. If the heterogeneous system must change on a yearly cycle the entire set of heterogeneous computers may have to be remapped to take up the slack of one obsolete specialized computer in the system. The long term life cycle support problem is extremely daunting. Keeping a mix of specialized computers and nonportable codes in the face of frequent obsolescence is against good software engineering practices.

The multiserver system provides a wide range of specialized services requested by users to reach their solution. Coordination of these operations, which is moving staged results from one computer to another with special capability, is an established method. Commercial products provide this capability. ISIS and TaskBroker are examples. The implementation remains a difficult one but the technology is understood.

Some researchers believe that heterogeneous computing could speed up a single computational intensive execution. In this form of processing, the programmer splits a large computation into stages. Each of these splits, matches one or more specialized architectures. Perhaps some of these splits are types of computation that expand with an increase in grain. For those cases, the communications fraction falls as the problem's computation expands. The problem is that when one component of the application execution does not reduce its computation fraction as it expands. A heterogeneous system would use an architecture with high performance multiprocessors (capacity-scalable) for the structure solution and a SPMD

rated multicomputer (cost-scalable) for the CFD computation. The results at the information surface between the fluid and the structure passes back and forth between the two computers on each iteration.

The chances of success of the heterogeneous computing approach depend upon the application. Extremely high communications rates between processor types are necessary. All information about the surface must go out, change to match the other architecture, and come in again on each iteration. In addition, this method may lose the advantages of data-locality-of-reference in caches or local memories. It is feasible to demonstrate applications but the programming and life cycle expense makes successful heterogeneous systems of this type doubtful.

4.2.3 Heterogeneous Life Cycle

In a heterogeneous computing environment the expense of specialized programming and upkeep of many different machines leads to high life cycle costs. Comparison of costs over the entire system life is necessary for a complete evaluation. A less complex environment with fewer machine types would have significantly lower development and long term support costs. The obsolescence of parallel computers exacerbates this problem.

In its present state, heterogeneous computing cannot create a paradigm shift among third party vendors because of the fragmentation and myriad architecture types. The performance differences and application suitability of cost-scalable multicomputers are not sufficiently different to justify buying several machine types. Government laboratory centers may find the use of heterogeneous computing effective on "grand challenge" applications because software labor costs and system support are not factors in the evaluation. However, the method is not presently suitable for working in the engineering process. There are several reasons:

- ◆ First, users will experience added wall clock time. The time saved would not be worth the extra effort.
- ◆ Second, the commercial user doesn't usually have the resources to use the complex heterogeneous methods.
- ◆ Thirdly, long term support is too complex and costly.

The methods and processes of parallel computer software engineering are very important to the building of heterogeneous computing systems. However, the present status of understanding of parallel systems is not adequate for effective use of heterogeneous computing techniques. The task of breaking apart a general application into heterogeneous parallel components is too difficult and too complex for quantitative design engineering. Each system with different networks, different loading, and different available architectures makes each site a unique problem. Until the problem of performance prediction for general applications on a single multiprocessor is solved the corresponding problem in a heterogeneous system should be shelved.

Performance of parallel systems in this environment is not adequately predictable. One cause is the dynamic behavior of C³I systems. The analysis of complex networks that include

parallel computer subsystems is intractable. Bandwidth demands on the system are not predictable and are nonlinear. As a result heterogeneous computing is not ready for C³I systems.

4.3 Multicomputers & Multiprocessors

Success of parallel computing in the commercial marketplace requires a virtual machine model that separates programming from architecture. Many diverse viewpoints and special interests now prevent the introduction of such a model. The necessity to classify the architectures is an indication that the "buying public" lacks an adequate depth of understanding about parallel computers

Distributed multicomputers are computers that "cost-scale." Cost-scaling computers have reduced bandwidth capacity as machine size grows. Each node has a fixed bandwidth per processor module added to the system. Multicomputers have a higher performance backplane interconnection that provides the capability to add additional processor nodes. These architectures are also cost-scalable because the addition of a module (unit of processor(s)) adds a fixed amount of bandwidth capacity to the system, theoretically at a fixed cost. Therefore, cost is directly proportional to the number of processor(s). In theory, by doubling the size, one only doubles the price.

Symmetric multiprocessors (SMP), are multiprocessors with limited processor count. Other multiprocessors have increased bandwidth capacity per processor module added. This gives approximately constant bandwidth capacity between any arbitrary processor for any size machine. Types could be local memory based (LMMP) or directory based shared memory based multiprocessors (DMMP). Other multiprocessors have a backplane interconnection that provides the increased bandwidth per processor module. The increase is that necessary for approximately constant bandwidth capacity between any arbitrary processor pair for any size machine. Doubling the size requires paying the costs of extra bandwidth necessary for maintaining the fixed bandwidth between arbitrary pairs. Multiprocessors, except for the SMP ones, are capacity-scalable. Symmetric multiprocessors typically have an upper limit based on the bus. Some allow added bus interconnect capacity at fixed processor counts. For example, a Cray SuperServer with sixteen processors has a single bus and the sixty-four processor version has a shared memory with four parallel busses. The capacity growth has a modular step of sixteen processors, making it capacity-scalable in four fixed increments.

4.3.1 Examples of Multicomputers and Multiprocessors

These classifications are theoretical. Few designs are pure cost-scalable or capacity-scalable. Practical hardware designs include modularity and a fixed upper limit to size. A multiprocessor or multicomputer becomes equivalent to a cluster when communication protocols require communications grain so large as to make the bandwidth irrelevant. A network of computers on a backplane is the result. Furthermore, a given design must suit the needs of many clients. To be effective in the marketplace, parallel products must provide a

wide range of input/output, disk units, memory sizes and speeds, and standard interfaces. In addition, one might expand the classification to include fault-tolerance/availability, and programmability/performance requirements. An example of meeting programmability requirements is the AT&T products. They provided either shared memory or directory-based cache memory coherence. The IBM SP2 and Meiko Computing Surface use memory mapped hardware mechanisms for synchronization and communications that allow effective passing of packets between processors. Table 4 - 1 gives the author's classification of some well-known machines.

TABLE 4 - 1 Classification of Parallel Machines

Machine	Vendor	General Type	Comment
Paragon	Intel	Multicomputer	Message Passing - one node processor dedicated to messages
CM-5	Thinking Machines	Multicomputer/ Multiprocessor	SIMD and single program multiple data message passing
SP-2	IBM	Multiprocessor	Memory Mapped Shared Buffer to Cross bar
3600	AT&T (NCR)	Multiprocessor	Memory Mapped - Cross bar

The discussion on multicomputers and multiprocessors will continue in the architectural trend discussion. Networks of computers are not suitable for the consistent achievement of predictable time response on general computations.

With local memory multiprocessors that are capacity-scalable, applications may increase in size as the processor size increases. However, those machines grow in cost since added bandwidth capacity per processor maintains the balance between application demand and processor size. The tradeoff of hardware cost against system software development and software maintenance over a long life is, however, seldom applied. Considering the cost of development of operating systems and tools, they compare more favorably with the costs of multicomputers. Including the costs of software for system integration, application development, and maintenance the multiprocessor often is more cost effective than a multicomputer. There are specialized applications where the multicomputer is more cost effective because the application matches the processor. There are not enough of those applications to gain an economy of scale. For the multiprocessor software engineering processes, portable development tools, portable application codes and programming model standards are feasible. The developers of software should choose target architectures based on the ease of software development and portability among their leading selection criteria. (*Availability and Delivered Performance* are the other primary criteria.) This has not been the case for many parallel projects.

4.3.2 Symmetric Multiprocessors

This small processor count type is important in the marketplace. The symmetric multiprocessor or SMP is now popular in client-server environments and is displacing mainframes in some important multiuser applications. SMPs are bus based machines with relatively low maximum processor counts. Obtaining good performance on these machines requires good locality-of-reference. Therefore, cache consistency mechanisms are provided in hardware. The programming model assumes equal shared access to memory from any processor. The builders of these machines assume that their maximum size limitation can be overcome by distribution of an application over a network or that a larger scale multiprocessor can be used to extend the range of processing.

Symmetric multiprocessors (SMP) typically use buses and are not capacity scalable multiprocessors. A bus is a fixed resource. Therefore, changing the number of processors changes each node's share of the fixed bandwidth capacity. Increasing the size of an application typically increases its communication demands. Therefore, an application can only scale in size if its portion of communication per size goes down. However, within those limits, the operation is less brittle than on distributed multicomputers. Programming is very portable since there is no mapping necessary. However, shared memory programming uses semaphore and mutual exclusion operations to coordinate memory access. Those approaches are error prone and difficult to program. A needed element for shared memory programming is a model for programming which allows symmetric multiprocessing code to be source-compatible with local memory based multiprocessors. A common set of agreed upon standard that bridges between cost-scalable multicomputers and shared memory multiprocessors is necessary. The Nexus specification (Aerospace Corporation) is possibly an example of such a message passing system. If used with only small packets it may be suitable for both symmetric multiprocessor and local memory based multiprocessor standard for communications.

4.3.3 Multiserver Systems

Workstations and their graphical user interfaces have brought the user closer to the computation. Putting the user at the display has significant advantages. The entire process of application set up, computing, and results analysis works directly with the user's immediate environment. However, if the computational stage of the engineering process is too long, the close coupling of the user to the application is lost. Keeping this coupling requires a link to an external computational engine. The files and applications that other members of a team also use, are common in an external file or data base computer. High performance local networks now provide this capability. The external compute and file engine is a server and the user station is a client. Clients and servers connect via networks.

4.4 Architectural Trends

4.4.1 General High Performance Architectural Trends

The trend in parallel systems architecture is toward more useful production computers instead of research oriented computers. ARPA funded massively parallel research computers have recently increased their interconnect structure capability. The commercial symmetric multiprocessor servers have increased bus bandwidth capacity and the number of processors. However, at the high end, shared memory vector supercomputers still set the standard for production supercomputing. They provide extremely high individual processor performance and a processor count of sixteen. Most often, they deliver a higher fraction of their maximum performance than other parallel architectures. The "massively" parallel vendors move toward production systems by lowering their maximum processor count and paying more attention to provisions for input/output and multiuser access. As a result they use the label "scalable" parallel instead of the massively parallel label, popular in prior generations. Vendors of scalable parallel system now place greater importance on system robustness to make their move to production computing. Throughput computing, which is serving multiple users for small transactions or replicated copies of a program, is a demonstrated capability of large scale parallel computers. Shared memory symmetric multiprocessors with low processor counts are very successful commercially as transaction processing throughput servers. Workstation and personal computer desktop vendors lead this change.

Emitter-coupled-logic (ECL) processors set the production standard for high-performance, production, technical computing. However, C³I systems have different requirements from "technical" production computing. Both physical and cost constraints limit the use of ECL based vector supercomputers in C³I systems. In addition, application of CMOS technology computers to C³I systems includes a combination of many types of processing that are not important to technical research computing. Commodity microprocessor architectures are better matches to the requirements of C³I systems. Therefore, we restrict this assessment to Complementary Metal Oxide Semiconductor (CMOS) based parallel architectures. We include both shared memory and "scalable" parallel multiprocessors in the architectural assessment since C³I systems require a range of machine sizes and types.

The shift toward production from research is leading to the following technical trends in parallel computers:

- ◆ use of workstation-compatible microprocessors
- ◆ convergence of processor count
- ◆ increased interprocessor bandwidth/ decreased latency
- ◆ programmability enhancing mechanisms
- ◆ proposed standards for message passing
- ◆ increased and more flexible input/output, disk, and memory subsystems
- ◆ increased fault tolerance, reliability, and decreased repair effort and time

4.4.1.1 Microprocessor Trend

Vendors of parallel machines use the same microprocessors commonly found in workstations. Workstation vendors are rapidly improving the processes and the designs of microprocessors to attain high performance. Using those microprocessors as the processor in parallel systems keeps the vendor on the latest technology curve at a low cost. Vendors also may not have to develop node operational software. Application vendors are also more attracted to extend for parallel ports than to do a total porting. Microprocessors with 64 and 32-bit data paths and 32-bit address ranges are now common.

Microprocessor vendors are taking a more active role in symmetric multiprocessor designs as shown in Table 4 - 2.

TABLE 4 - 2 Microprocessor Application to Parallel Computers				
Microprocessor	Parallel Computer	Type	Interconnect Structure	Comment
MIPS	SGI PowerChallenge	Multiprocessor	Bus based Symmetric shared memory	(18 most powerful nodes or 36 with less powerful)
PowerPC	IBM SP-2	Multiprocessor	Cross Bar	Large processor counts
Alpha	Cray Research T3D	Multicomputer	(3-D mesh)	Tightly Coupled to Vector Super Computer
Alpha	Digital Equipment Corp	Multiprocessor	Bus based Symmetric shared memory	
Precision	Convex Computer (Exemplar)	Multiprocessor	Both Bus Based Symmetric and SCI directory based shared memory	
SPARC	Sun Microsystems and Cray Research	Multiprocessor	Bus and Multi-Bus Based Symmetric shared memory	(Sun 20 and Cray 64 maximum nodes)
SPARC	Thinking Machines	Multiprocessor	(a limited fat tree)	SPMD and SIMD operation

Microprocessor	Parallel Computer	Type	Interconnect Structure	Comment
Pentium	AT&T (NCR)	Multiprocessors (2 types)	Symmetric shared memory to 32 nodes, Crossbar based above	Software tools across both architecture types (OLTP and DBMS)
i860 (not a workstation microprocessor)	Intel	Multicomputer	2-D mesh	Large maximum count
Pentium	Compaq, Encore, Gateway, Dell, IBM, AST Research, Hewlett-Packard, etc	Multiprocessor	Small Processor Count Symmetric shared memory for Network Servers	UNIX and WindowsNT Bus Chip sets: Corollary, LSI Logic, Pequr and Wyse
in-house	nCUBE	Multicomputer	(hypercube)	Large maximum processor counts

4.4.2 Maximum Processor Count Trend

Scaling of an architecture to Teraops (10^{12} operations per second) peak performance is a feature necessary for only a few systems in a few locations. Massively parallel processing (MPP) versions of cost-scalable multicomputers sacrifice volume sales at lower processor counts. Scalability over a range of ten to a few hundred processors is very important to attain an adequate volume to support the gross margin necessary for building operating system and other system software. Sales volumes necessary to achieve critical economic mass require volume at lower processor counts and demonstration of high processor count capability. The trend in "cost-scalable" multicomputer architecture is to meet this range of processor counts with the most powerful node processor available. The latest architectures in the parallel market trend toward reduced maximum processor counts. Recent entries from IBM, Cray Research, and Convex are beginning at smaller maximum counts and with plans to extend upward.

Symmetric Multiprocessors

A second trend that has significantly more economic power behind it is following this sequence:

- ◆ to start at a very low count, as low as two to four
- ◆ convince users of its value
- ◆ increase the count over time

The vendors of workstations and personal computers have taken that approach. Both Sun Microsystems and Compaq sell over 10,000 symmetric multiprocessors each month for file, OLTP, data base servers and multiuser servers. Sales of these systems were almost \$5 billion in 1993. IDC

projects a business level of over \$7 billion in 1996. Profit margins are higher than in single processor systems. As a result, a significant fraction of a vendor's profit comes from multiprocessor server systems. Thirty percent of Compaq's profits came from servers with 3 percent of the units sold. The data shown in Table 4 - 2 shows that competitive designs are coming from symmetric multiprocessor vendors using high performance bus systems. These processor types have the opposite trend in processor count from the technical computing multicomputers and multiprocessors. Once limited to four to eight processors, symmetric multiprocessors now go to sixty-four processors (Cray Research SuperServer) using four complete cache coherence busses. AT&T Global Information Systems (NCR) sells server systems with data base and OLTP applications. Their models scale across symmetric and tightly coupled distributed multiprocessor architectures and over the range of two to 512 processors.

4.4.3 Interconnect Latency and Bandwidth Trend

The cost-scalable multicomputer vendors trend to the optimum point on the connectivity (the number of network routing paths from a given point) and bandwidth curve. Now under the name of "scalable" parallel processing, this trend is changing. A generation back, researchers favored the hypercube and other high connectivity networks. Then designers began to reduce the connectivity and increase the bandwidth of individual links. Their reason was that the communication mechanism - message passing using large messages - fits well with low connectivity interconnects. Delays in the protocols for creating and transferring messages dominated the latency. For example, machines from Intel changed connectivity from hypercube to a two dimensional mesh or torus. Due to protocol processing delay, the message latency of the interconnect was less important than the time to transfer the message body. Engineers are reversing the trend by reducing protocol delays and shortening messages. Examples of the trend reversal are IBM with eight way cross bars, and Cray Research with a 3-D mesh. These changes stress low latency for small packet transfers. nCUBE's use of an eighteen-way path is useful in the multimedia server application, another indication of a rebound in this trend..

4.4.4 Obsolescence Trend

There is a flux in performance and architectural designs on the market. From the prior generation of multicomputer machines (MPP) to the present one (SPP) Intel, Thinking Machines, nCUBE, and MassPar have not succeeded in creating a rapid growth in new applications. These early machines had a scarce list of features. They lacked virtual memory. Their designs targeted doing well on common functions and benchmark cases. Performance on complete applications was disappointing. They used inadequately robust interconnect designs. Most had inadequate input/output capability for production environments. They concentrated on research markets instead of commercial and industrial technical applications. These machines emphasized peak performance instead of the reliability and programmability necessary of the commercial marketplace. As a result, a large government research component in the market is necessary for their existence. The risk to the government is that many programs committed to those machines cannot take advantage of software from a broad base of suppliers. In addition, the general needs of C³I systems lack suitable parallel hardware and software because the MPP machines are suitable for only a limited set of application types. C³I systems builders also cannot take advantage of a broadly accepted programming model and tool sets that meet all their needs.

The difference in the research and production oriented machines shows up in results on the National Aerospace Simulator (NAS) benchmarks. The IBM (SP1) and Cray (T3D) products are consistently above those from the earlier generation. The importance of high bandwidth, low latency and high connectivity shows in the results reported by Cray Research on the NAS benchmarks. The table of results are given in the application essay in Table 2 - 4.

4.4.5 Programming Model Support Trend

Features have been added to the interconnect/backplane to support either shared memory or scalable parallel message passing communications between processors. The shared memory programming models have cache coherence components that allow cache updates when any processor writes a location. Hardware coprocessors now directly support the message passing model, for example, the Intel Paragon. Dedicated hardware that transfers buffers between processor memory (e.g., IBM SP2) also directly supports message passing. Meiko supports the Computing Surface model in a VLSI interface circuit for each processor interface to the interconnect. AT&T (formerly NCR) provides its commercial software tool for decision support DBMS access on small symmetric multiprocessors and on its larger local memory based parallel machines. The larger of these machines emulate shared memory to ease programming effort.

Meiko, IBM, Cray and Convex have recently introduced machines with global or virtual memory, more robust and reliable interconnect designs and more balanced input/output capability. IBM, Cray, and Convex have made heavy investments in complete solutions, including commercial business and production technical design and engineering applications from independent software vendors. They, by necessity, have concentrated on higher reliability and programmability than the early research machines. If they succeed in meeting commercial availability and programmability standards, they have the potential for widening the acceptance of parallel computers. This could convince large numbers of independent software vendors begin to provide new parallel applications.

The shared memory programming model is to update the locations held in a cache whenever any processor writes to the location. This capability is "cache coherence." Cache coherence is found in machines ranging from small two and four processors network servers to several hundred processors in the AT&T 3600. The Stanford Dash computer uses directory-based cache memory coherence and common microprocessors. The Scalable Coherent Interface (SCI) bus used by Convex also has a large processor-count with directory based cache coherence.

For message passing the mechanisms are different in concept because the architecture has a wider range of update and transfer capability beyond the cache line. For example, the Intel Paragon has an i860 dedicated to providing message passing and routing through its 2-D interconnect. The memory-mapped buffer appears in a cross-bar based interconnect structure. Theoretically, it provides the capability for transferring small messages at low latency. The IBM SP2 has a memory-mapped buffer for transfers. Special circuits take data from the buffer and deliver it from node memory to node memory. Similar support is found in the Meiko Computing Surface and the AT&T 3600. The Cray T3D shares these capability and performance features.

4.4.6 Capability Trend

Significant increases in input/output capability and disk support makes these machines practical for continuous operation. For example, the Cray Research T3D has a significant advantage due to its proven input/output system. Cray's T3D uses an attached vector Supercomputer and its input/output and storage system interfaces. The AT&T 3600 has disk drives and communication ports at each processor. The Thinking Machines CM5 has a very high bandwidth between its disk drives and its processors. As a result, advanced commercial information decision systems find that the AT&T 3600 and CM5 are ideal for large, complex queries. The AT&T 3600 runs Oracle, Sybase and a proprietary data base system. TMS's CM5 is typically used in SIMD mode for rapid sorts of information. The Oracle DBMS has been successfully ported to nCUBE, Meiko, and symmetric multiprocessors based on workstation microprocessors. A glowing success of parallel systems is the capability to provide high throughput for transaction processing systems. Pyramid, Tandem, AT&T, Sequent, and Encore are companies that do business in this important application area. AT&T hardware sales for this purpose exceeded all the MPP vendor sales in 1992 and in 1993. However, vendors have recently added the capability for providing quick responses for complex queries. They accomplish this by a preprocessor parallel program that breaks apart transactions for concurrent execution.

4.4.7 Utility Trend

The latest trend is to add features for increased availability. AT&T 3600, with folded Banyan interconnect, has built in redundancy control for fault tolerance. The folding provides lower latency when fully functional. One failing of research oriented parallel computers is that both hardware and operating system reliability have fallen short of that required in both engineering and commercial applications. Since common use is outside a mission critical application this deficiency survives. The trend is to correct this shortfall. Adaptation outside the research computing environment requires high availability.

4.4.8 Cost Trend

Cost trends are up but with increased delivered performance. This comes from the necessity to increase interconnect performance, provide fault tolerance, develop operating systems on new architectures, and provide the input/output balance necessary for out-of-the-laboratory operation. System software costs have been significant. Reliability and instability problems have plagued some scalable architectures. The conclusion is that good computers are expensive to build.

The foundry cost differential between using a specialized processor instead of a standard workstation microprocessor is not the significant cost benefit factor once believed. Interconnect costs are necessarily high when vendors serve a broad application spectrum. Lack of support for a broad application set restricts the market and doesn't allow for amortization of development costs across several machines.

4.4.9 Real Time Trends

The capability to perform acceptably within hard real time systems is not one that scalable parallel computers have reached. This area remains a specialized parallel processor field with corresponding high life cycle costs and long term support issues.

4.5 Architectural Trend Conclusion

Trends toward more robust parallel computer systems are turning in the right direction for the software engineer. A common programming model requires robust hardware and operating systems. However, the capability to perform on complex and dynamic applications is not yet a capability of available machines.

The latest introductions have punctured the idea that cost-scalable computing is generally useful. Capability-scalable machines have a better cost to effective performance ratio than cost-scalable ones. The extra investment in interconnect structure is off-set by lower software costs and wider applications set and market. The extra hardware and system software for fault tolerance are necessary for the commercial market. Costs in that area are returned by satisfied clients. Providing compatible software tools over the range of processor counts and varying the architecture also pays off, as shown by the sales success of AT&T for its OLTP and DBMS machines. As capability-scalable machines grow in the commercial market the benefits of large scale markets will provide better tools and broader understanding. This doesn't mean that these trends will solve all the problems of parallel programming. It only means that it is now feasible to accomplish some ideas and tools necessary for portability and low-risk design.

Machines are now more capable but with fewer processors and lower peak speeds. Higher and more stable communication bandwidth capacity between processors is the result. Vendors are putting hardware support parallel commands and programming models into practice. Availability increases because fault tolerance features must meet commercial user needs. Some data base and transaction type applications have been ported across a range of symmetric multiprocessors and distributed multiprocessors. The mix of architectures is still too diverse. However, the trend to increase interconnect capability reduces the wide differences between architectures. These differences become less a disadvantage to software engineering processes for parallel systems because with adequate capability the timing of parallel instructions over a range of typical example applications can be used to evaluate the suitability of a machine for an application. In that case, the architecture's structure can be hidden and a quantitative evaluation of cost to performance benefit can be calculated from the parallelism content of the application.

Parallel Software Engineering Assessment

5 Market Forces and Issues

5.1 Commercial-off-the-shelf (COTS) Parallel Computers

The present research parallel computers are not adequately robust in either interconnect performance or machine availability for commercial viability. COTS approaches presently succeed for workstation C³I components, but not for parallel computers. C³I system builders who need parallel computing need a strategy to leverage COTS advantages and encourage systems with commercial viability and widespread acceptance.

5.1.1 Strategy for Future Cost Effectiveness

Commercial-off-the-shelf (COTS) components are important and provide an available technology for building high performance C³I systems. The economy of scale of having commercially available hardware leads to widespread use of COTS workstations and associated technology. Operating systems, data base systems and high level languages are readily available for scientific and commercial uses of workstations and personal computers. Standard committees have captured the most common workstation operating system (UNIX) and created a military standard (POSIX). Windows NT from Microsoft is expected to increase the market for symmetric multiprocessor (SMP) processors in commercial client-server markets. The success of workstations and their ever increasing performance and programmability leads to the same high expectations for parallel systems.

The strategy is:

- ◆ encouraging the parallel industry,
- ◆ nurturing it into wide acceptance
- ◆ reaping the benefits of associated commercially developed software and hardware

These strategies are valid for workstations and personal computers. The same strategy should work for parallel computers but has not.

One reason the strategy has not worked is that advocates treat COTS scientific parallel high performance computer systems as specialized processors and not as general computing products. Before the COTS strategy, designers developed parallel computers into C³I systems specifically for a given need. Each had its applications carefully mapped to achieve high performance. Designers isolated these specific capabilities. They placed them only where needed and where structure matched needs. These special-purpose, parallel computers had specific, well-engineered capabilities. The life cycle costs were high (each required special maintenance) but the system met performance and physical constraint goals.

5.1.2 Research Viewpoints

Presently researchers treat COTS systems in much the same fashion: they are fielded in only a few sites and each is programmed for a set of specialized applications. These applications are typically carefully selected to match the limitations of the parallel computer. In the original approach the parallel computer was carefully selected to match the C³I needs. This approach provides some advantage in that a few different COTS systems share the same architectures. However, the cost is loss of efficiency, performance and capability because system needs came second. The number of parallel computers sold remains small because there is no broad applicability or market for the parallel computer.

5.1.3 Building Industry Capability

This is not a good strategy for attracting and building a COTS industry. First, independent software vendors find less than their critical mass since there are only a few tens or hundreds of each model. They amortize their application software development costs over thousands of sales to main sound business practices. Second, hardware vendors fail to achieve promised hardware cost saving. They failed to consider development, marketing, and financing in setting their cost goals. The vendors did not consider that they must amortize their hardware interconnect, mass storage, operating system design and development over only a few sales. Third, the vendor finds that hardware is obsolete by the time the operating systems development completes. (NASA found that one well-known cost-scalable multicomputer system failed on the average of three times out of twenty-four tries of the same application execution.) Fourth, the reputation of parallel computers suffers due to the difficulty in attaining desired performance and functionality. Finally, they do not meet the wide range of complex and demanding mix of execution types expected in either a C³I or large-scale, commercial, production system. A large market requires the general capability for meeting real time, numeric, symbolic, visual, control, etc. processing types.

The COTS approach offers the advantages of common and widely used tools, software, computers, and peripherals. However, for scientific high performance computer markets there are no commonly agreed upon common architecture models. Each vendor must bare the development and maintenance burden of the underlying operating system software and languages. Hardware vendors need to leverage the efforts of independent software vendors to reach the critical mass market.

5.1.4 Failure at Generality

The failure at generality of the systems forces an interaction between hardware and software that leads to nonportable codes. A key feature of the success of workstation software is that applications could be easily ported between vendors without regard for specific hardware machine details. Hardware vendors do not need knowledge of the software vendor's applications in any specific way and vice versa. Each goes on with assurance that they can attain a reasonable performance level without tying themselves to a specific matching technology. Without independence of hardware and software developer there cannot be a successful COTS strategy for parallel computing.

5.2 Opportunity and Challenge

The challenge to the C3I system builder is to encourage a market for parallel COTS products. To attain a market size suitable for attracting the independent software vendors the hardware products must be adequate for a wide range of applications. The software must be portable with reasonable performance across a wide range of architectures and machine sizes. This results in a sacrifice due to increased hardware costs because the vendor must beef up the interconnect and improve fault tolerance and recovery capability. Robust interconnect performance and machine availability is necessary in the larger market. It also means that a significant increase in portability and programmability is gained when buyers force these capabilities on machine vendors.

These changes offer new opportunities for enhancing command, control, communications and intelligence (C3I) systems. Market-coupled, commercial-off-the-shelf (COTS) solutions allow C3I builders to meet their budgets. Application specific parallel processors necessary for meeting tight physical constraints could use the same common model for efficiencies in software development. The Air Force needs a strategy for making effective use of technology and market driven COTS technology.

A part of that strategy is to encourage the software engineering tool set that is necessary for the commercial successes and extend the marketplace successes to suit C3I needs. In addition, success of a programming model applicable to the expected commercially successful architectures would serve the C3I builder. The trend in production technical computing is toward data base and object-oriented storage for design and analysis information. That trend turns the computing attention center away from Fortran applications, and toward use of application packages and system integration languages like C and C++. Most commercial research and engineering production already use applications from independent software vendors. The software that is written is for integration of the packages to make the engineer's job easier. Industrial production centers use that same approach, mirroring the purchase of software vendor and integrator packages.

5.3 Market Driven Technology Forces

A successful strategy for C3I system and software engineering must be consistent with and take advantage of the market dynamics of commercial information and technical engineering systems. In this essay, we make a case that the growth in small symmetric multiprocessor systems might leverage parallel software engineering processes. Use of a common programming model allows these processes to also work for low-latency, scalable multiprocessors.

Competition ties technology advances and market forces together. The computer marketplace is an environment of intense competition, resulting in rapid technology advances and changes. These changes are generally beneficial to the users - performance and capacity grow significantly each year at constant cost. Often performance increases cover up a product or system shortfall. Marketplace dynamics in parallel processing brings the opposite. The changes increase the disadvantages of hardware obsolescence, disappearance of support, and

a dearth of supporting software tools. Because of this flux, technology advances may even exacerbate any weakness in the software development process and management of large scale software efforts.

Today's dynamic changes in the computer market result from the displacement of Emitter Coupled Logic (ECL) processors by Complementary Metal Oxide Semiconductor (CMOS) Logic microprocessors. The result is a rapid advancement in speed of the single microprocessor chips. Low cost and open systems standards make this technology useful as a high throughput (many independent operations per second) computer for workstations and parallel computers. For structured applications with fixed data locations few processors (10 to 30) reach a single processor's (ECL) operational speeds with lower hardware cost. Advocates think that cost-scalable multicomputers built from very high performance parallel CMOS computers will far exceed the capabilities of today's highest performance ECL computers. They also hope that they will be available at a reduced cost per unit of performance.

Large volume production of microprocessors is necessary to recover development and foundry costs and to allow a margin for continued development. The most popular models will rapidly evolve and improve while the less popular ones become obsolete and disappear from the market. The Intel i860 is an example of volume inadequacy and architectural performance limits, causing obsolescence. While the Intel 80x86 / Pentium series is a successful volume case. A processor applications base and choices of operating systems are critical factors for market success in personal computers and workstations that use microprocessors.

Other equally dynamic changes in the paradigms of computing are underway. The human interface is changing rapidly. Object oriented screen building has reduced the cost of conventional interface construction. Voice commands, pen control, written text to ASCII character transformation, and wireless and remote, self-contained information collection capabilities are now available. Network control and communication's software for building homogeneous clusters of computers is now feasible for throughput demanding applications. Forecasters expect that wide-area communication speeds will increase and become more broadly available. They project that future transfer speeds will be greater than today's workstation microprocessors even at their higher operational speeds. They will not be able to do the necessary protocol processing in real time without extra hardware assistance. Technology advances in field programmable devices, magnetic storage media, and virtual reality add to the opportunity for computer technology's capability to contribute to C³I system requirements.

In parallel systems the microprocessor cost is not the critical issue - many highest performance microprocessors are available for \$500 per package, while multicomputer nodes are priced at 100 times that cost. Even the design and foundry costs of a specialized microprocessor are not overwhelming if foundries are available. The critical factors are the development cost of a robust system, compiler software, operating systems and the existence of an application's base. Achieving that base requires the investment of many independent software application developers. Broad market acceptance of a microprocessor is necessary for attracting the investment in operating system, languages and applications. For example, IBM, Motorola, and Apple combined forces for ensuring a broader market, adequate volume

and applications base for the PowerPC microprocessor. HP and Intel jointly announced an advanced processor to replace the Pentium series.

5.3.1 Demand for Commercial Parallel Systems

A recent New York Times best seller, "Reengineering the Corporation," by Hammer and Champy, describes a revolution occurring in corporate organization. Corporations are rethinking fundamentals and making radical changes to achieve dramatic results. The key to that revolution is the identification of "value added processes." Hammer and Champy emphatically make the case that all large United States corporations must undergo this "Reengineering" if they are to remain in the worldwide, competitive marketplace. Many major US corporations are committed to Reengineering. The key to success is to use information, simulation, communication, computer information technology as the facilitator. The level and direction of the "Reengineering" depend upon available and projected information technology. A significant market is emerging for parallel computers to support the needs of corporate Reengineering.

These dynamic and fast-moving trends create the market for new multiprocessor software products. These new products create the conditions for additional hardware and software products. Synergism like this creates a paradigm shift. Including the larger scalable multiprocessor systems in this paradigm shift requires a programming model that allows them to execute the same application's code. The technology necessary to accomplish this is twofold:

- ◆ a combination of adequate low latency and a high bandwidth from hardware
- ◆ a "virtual machine" model that works on both smaller symmetric multiprocessors and local-memory-based, bandwidth-scalable multiprocessors

A factor that helps the process is that the needs of commercial and industrial complexes and C³I system ones are sometimes similar.

5.3.2 Acceptance of Parallel Processing in the Marketplace

Parallel computing also requires broad acceptance of both processor usage and parallel architecture. Some successful forms are emerging in the marketplace. C³I parallel systems must leverage off those successes and extend them for their military use.

Market forecasters predict sales of several hundred thousand symmetric multiprocessor servers for calendar year 1994. These may average only four processors per server. They are based on CMOS/RISC microprocessors with cache coherence shared memory control chips. Their success is based on significantly reducing costs below the equivalent mainframe technologies. SMP machines are commercially available for Pentium (AT&T, Encore, Compaq, Acer, etc.), SPARC (Sun Microsystems and Cray Research), Precision (HP and Convex), MIPS (SGI), and Alpha (DEC).

For high-throughput on-line transaction processing and data base access, the industry believes in "shared nothing" local memory based multiprocessors (LMMP). Most models of these are capacity-scalable multiprocessors with increased interconnect structure to maintain performance across a large range of processor counts. They are found to be effective in both performance and price on multiuser on-line transaction and data base applications. They are now supplementing mainframes in some large new information systems. Data storage servers also have a large market. These machines are inexpensive compared to mainframes, however, they are robust and have high performance interconnects. Such structure is expensive to engineer and produce. Successful market entries have paid close attention to availability, programmability, fault tolerance and Input/Output capability. Capacity scalable LMMP machines are available from IBM, AT&T, nCUBE and Meiko. The LMMP machine from Thinking Machines is not fully capacity-scalable as the "fat tree" capacity favors local communications. The LMMP systems from Intel, the Paragon, is a cost-scalable multicomputer with poor multiuser capability. In addition, Intel uses a special purpose microprocessor, the i860. Intel sells it also for signal processing applications. The others use large sales volume microprocessors also found in workstations. IBM uses the Power2 in its SP2 parallel computer. The Power2 is a multiple chip module processor provided in their top-end workstations.

Cray Research provides its T3D for large scale applications with multiple Alpha processors, and the SuperServer with multiple SPARC processors. The T3D has a very rapid access to memory. Note that AT&T and Cray Research bridge the shared and local memory barriers and provide machines in both SMP and LMMP categories. Convex provides an SMP programming model via bus based systems at low processor counts and a directory based coherent cache (DMMP) system using the Scalar Coherent Interconnect (SCI) standard. Directory-based coherence preserves the shared memory programming model to larger processor counts. Memory is local but updated via separate mechanisms. Convex provides one using the directory coherence features of the SCI bus. This trend points to the possibility of shared memory programming on larger processor count system.

Because of the recognition of their effectiveness, industry routinely uses high performance parallel computing for OLTP and DBMS servers over a range of sizes and architectures. Data base and transaction applications for parallel computers have been ported and are available from several independent software vendors. These products span both symmetric multiprocessor and local memory based multiprocessor architectures. An example is Sybase DBMS on the AT&T Global Information System product line with both SMP and LMMP architectures. The success in this field shows that if parallel machines are adequate and useful for an application that independent software vendors will respond. C³I systems could use this type of processor for applications that match them. The suitable applications are multiuser transactions, data base storage, and replicated copies of programs, one copy per user. AT&T's robust interconnect structure in its model 3600 has proven reliability. More important, the operating systems for the commercial system are robust and reliable.

5.3.3 Technical Computing Marketplace

Scientific research, high-performance parallel computers have not achieved the level of effectiveness necessary for broad acceptance. Although highly popular for computer science and algorithm mapping research funded by the government they remain outside the production for day-to-day operations.

Computer science research has failed to develop in a direction that fits it into the production engineering process. A concentration of grand challenges has lead the field away from the industrial processes and operations necessary for market acceptance. Typical engineering projects are performed using workstations and low processor count servers reached via a network in a client-server mode. The server might be a four-processor one that provides files for several users. Much of this parallel processing is now limited to replicated applications for different parameter sets. Unfortunately, a single large application, such as simulation or detailed operations of an engineering process, gets little speedup gain from the present server technology. Often engineers wait overnight for critical runs because the necessary processing performance is not available. If the four processor servers delivered speed on a variety of engineering processes a sixteen-processor system would be justifiable in these environments. Instead of concentrating on creating an integrated industry, parallel programming research has left these practical engineering process applications and concentrated on applications that require Teraops level processing speeds.

5.3.4 Commercial Parallel Systems

The present research parallel systems do not meet all the requirements of the production center for technical or commercial computing. Programming models are not portable and require architectural knowledge and detail level programming. Advanced commercial efforts for decision support systems require development teams of tens and hundreds of people. The lack of software engineering processes to deal with system and parallel computers often leads to these inefficient projects. Much of the present parallel computer research community considers scalability of hardware above that of software.

However, symmetric multiprocessor applications establish a basis for independent software vendor products. Symmetric multiprocessors provide a few primitive mechanisms (e.g., threads coordinating via locks) to allow the programmer to deal with concurrency. The thread programming model is often not portable with other vendor's hardware and is not portable to local memory based multiprocessors. Sales of these machines, data base and transaction processing software, and a full spectrum of business process applications is now available on those machines. The popular client(a workstation) and server(a parallel processor) operation allows many corporations to spread their operations across the nation on wide area networks. The commercial software package industry has built a significant inventory of software, making skilled and trained personnel with experience available. C³I systems often require the similar transaction, data base, and reporting applications. C³I system designers should understand and capitalize upon these capabilities. Due to the small

processor count, these processors cannot solve some of the large scale problems of C³I systems. However, they can contribute a significant level to the matching application types.

This leaves the large scale LMMP processor. These systems are also finding applications in industry for support of decision support systems. All of the well-known examples communicate with an operating legacy system. The systems extract data and feed it into the parallel computers for processing. They work on large and complex queries and analysis applications. They often cut completion time down from weeks to days. However, there is a demand for even quicker results. The primary disadvantage is that each example is ad hoc and required a significant staff for development and operational support.

Large scale commercial applications have expectations of mainframe-like operations (general applicability) but at much higher performance. Multitasking (switching between multiple users to overlap and hide delays) makes interconnect latency less important to getting high throughput for transaction and data base applications. However, a single user with a large decision support application with complex queries requires multiple processors. AT&T provides software that breaks complex queries into independent parallel transactions for execution on either SMP or LMMP systems. This allows use of local memory based multiprocessors for complex queries. However, large scale parallel systems still do not have the general applicability of mainframes. The vendors of large scale parallel systems have "skimmed the cream" of transactions and data access. Their products do not provide the versatility of programming and effective operation provided by the supercomputer or mainframe on complex and dynamic applications.

Early examples of commercial systems that have embraced parallel methods, show that the present successful parallel offerings cannot replace entire systems. Large information systems have additional capability or function instead of simple replacement of "legacy" or "heritage" or "dusty deck" operational systems. Industry now uses parallel computing to accelerate processing of new throughput-oriented applications. However, the mainframe or supercomputer remains the operational, day-to-day production system for heritage or legacy applications. Often the prior software investment is too large for a parallel system to displace an operational system. The proprietary file systems upon which these systems have been based compound this problem. The investment is significant and requires the following:

- ◆ extract data from the legacy system
- ◆ insert it into the new parallel system
- ◆ do the decision processing
- ◆ extraction
- ◆ post analysis
- ◆ put data back into the legacy system

Often this effort requires tens and sometimes hundreds of people. Over the next five years both commercial and C³I systems will likely consist of mixed legacy and advanced technologies. Software engineering progress is necessary to build this mixed operational capability to allow incremental insertion of parallel technology.

5.4 Strategy for Embracing the Commercial Successes

The parallel computer industry is experiencing significant competition at all levels of systems. In the Intel 486/Pentium marketplace AT&T, Pyramid, Sequent, TriCord, Compaq, and Encore plus many other smaller vendors are already present. Compaq has already sold several hundred thousand multiprocessor systems. Intel recently included key parts of the shared memory consistency control within the Pentium microprocessor chip. In addition, Intel has announced a standard for Pentium systems that will allow the commodity "clone builders" to join the symmetric multiprocessing hardware market. Cyrix has announced a competing open standard to prevent Intel patents from cornering the market.

Workstation vendors have also joined the market for parallel servers. For example, in 1994 Sun Microsystems managers expect to reach 150,000 system sales for servers, although with an average of only four processors per server. Other workstation vendors (Hewlett Packard, Silicon Graphics, Digital Equipment Corporation, IBM, etc.) have powerful symmetric multiprocessing products. These systems are based on shared memory multiprocessor busses that provide cache control. Single bus versions of these systems allow up to twenty processors (Sun SPARCcenter2000). Cray Research is selling a sixty-four-processor SuperServer SPARC system using shared memory organized around four such busses.

This commercial trend reveals that hardware vendors are expecting rapid expansion in the use of symmetric multiprocessors. These computers operate well in a throughput mode where several users share a processor. These symmetric multiprocessors are not subject to message passing mapping effort. However, they require threads programming using locks and semaphores - a technique requiring very careful programming to avoid concurrency conflicts. Independent software vendors are not enthusiastic over use of proprietary threads. POSIX threads are not yet widely received because vendors are still pushing their own thread environments.

The opportunity for the Air Force is to recognize the shortfall in proprietary threads and large-message passing models. Then define a standard programming model that will work on both shared (SMP) and scalable (LMMP) architectures with low latency. One would not expect high effectiveness on clusters due to high protocol latency in protocol based parallel systems. Possibilities include carrying out a small-packet, message-passing mechanism in software for both symmetric multiprocessors and for the buffer exchange hardware for LMMP designs. Synchronization latency would be similar in both cases. Highly portable code results. With a similar latency and common programming model, applications are portable across a range of sizes and architectures.

Scalable parallel processing applications and symmetric multiprocessing now share the same software applications and tools market. Independent software vendors would find the opportunity for profit from parallel computing. This produces a larger base for the creation of software engineering tools and processes. The Air Force goal should be to influence the direction of a standard those tools so they also apply to large local memory-based multiprocessors required for dynamic, complex C³I systems. The result would be programmable scalable parallel software applications that span architecture types and sizes.

Parallel Software Engineering Assessment

6 Status of Software Engineering for Parallel Systems

6.1 The Software Engineering Goal for Parallel Computing

Separation of hardware and software development is important to the engineering of parallel hardware and parallel software. Without this capability each effort by hardware vendors and software developers is ad hoc and has little potential for reaching the large markets necessary for commercial advantage.

Parallel systems have the common problem with other complex engineering tasks because careful functional, performance, and resource modeling are required at each phase of developments. As a result software that must be mapped to a parallel architecture cannot be separated from the hardware adequately to ensure that design does not overwhelm the parallel computer. However, a well defined parallel instruction set should server as a equivalent to the gate construct. Such a mechanism would allow the same simulate at each level and an automated design process.

6.2 Process Mismatch

6.2.1 Limited Commercial Applications Successes

The reason that multicomputers are not ready for large scale decision systems in commerce is that they are too limited in application type. [Richmond: July 1993] The introduction of large processor count multicomputers and multiprocessors into commercial uses is inhibited by the following reasons.

- ◆ performance is uncertain
- ◆ difficulty of carrying out the project
- ◆ intensive interaction with heritage systems and interface is difficult
- ◆ software tools are not in place
- ◆ obsolescence leads to tactical applications
- ◆ high complexity leads to high life cycle costs
- ◆ cost of project raises visibility to high-level management
- ◆ economics allows only incremental change not radical ones

Yet the introduction of symmetric multiprocessors as data base management system and on-line-transaction-processing servers has been phenomenal. The success of Compaq and Sun Microsystems in the server market has led to others joining the market. This market is for two to four processor parallel computers. The conclusion is that successful applications lead to a large and high value added market if the technology is consistent with the system engineering process.

6.3 "Sequential" Software Engineering

System design and development processes and tools are now maturing. Applying them carefully provides a well-managed path to system development. The process of software engineering has evolved to provide a basis for tools and for managing the building of long-lived, complex C³I systems. Steady improvement by tool builders has provided a steadily improving progression of process environment tools for large system development using sequential computers. The results are Computer-aided-software-engineering (CASE) environments, built-in configuration management (SCCS), and reusable language features (from Ada and Object Oriented languages). Personal computer programming systems now feature some encapsulation and code reuse aspects of modularity and libraries. For example, Microsoft provides VisualBasic data and code encapsulation "objects." Their Dynamically Linked Libraries aid programmer productivity and maintainability. Builders of commercial products used some ideas that led the DOD to create the Ada language.

To avoid misunderstandings and errors in specification between developers and users the Spiral Method is available. The Spiral Development method allows the detailed user, developer and tester interaction at stages that lead to a final system. A progression of stages and builds lead to a finished system that more accurately meets users needs than a one-shot development. Tools for providing performance estimates for system designs (e.g., SES Object and SES Workbench) are now available. These tools let the designer apply them for both hardware and software at very detailed levels.

The success of graphical user interfaces lets the user create a new demand for higher performance. A paradigm of close user interaction results. The resulting cycle is setup, computation, and analysis with intimate user control. Often the developer's dilemma is that user expectations grow faster than performance. Technology influences those processes. This rapid change gives the user increased expectations during system development. Since new features become feasible as technology advances users expectations grow. Command capability provided by Graphical User Interfaces, specially the object oriented ones, let the user increase demands for features and performance. The computer industry has reached a state that users expect a holistic problem solution environment. This is where they interact with the system in short turnaround times for an entire solution process of preparation, computation, and analysis. (For example, VisiCalc and its modern descendent spreadsheets provide this create, compute, observe, and repeat.)

A "holistic" approach by human interaction to reach "non-canned" solutions is a leading example of growing expectations of processing responsiveness. Corporate Reengineering projects show that information systems give a "case worker" the capability to do an entire process. This approach is far more efficient than using several specialized workers. Instead this new model of processing is that the user gets results in adequate time to iterate the solution several times in a day. Thus, demand grows due to increased human interaction with the computer system. The capability to make informed decisions using interactive iteration is a demand brought by the human interface capabilities of workstations and personal computers. This demand exists in many professional projects: business decision making, software development, mission planning, computer-aided-engineering, and control systems.

A paradigm shift is needed to focus the parallel research community on extensive parallel applications and architectures that are capable of supporting them. Mapping the same problems on the latest new architecture has little learning or state-of-the-art advancement potential..

Programming tools for aiding development of parallel programs are widely available but little used. Academia fills research journals with new reports each year. However, parallel computer tools seldom reach the users outside research laboratories. One reason is that the user community for parallel processing is research oriented. This research orientation leads to building more tools instead of seeking a consensus of common tools and using them. The result is an availability of many tools, used only briefly and by a small portion of users. A second reason is that there is a lack of a common theme and principle. Too often, tools are reworks of already tried ideas carried out on the latest architecture. Each generation goes to the same depth of understanding. They often fail to advance general principles. Mapping production applications to parallel computers is difficult. Production applications are dynamic, complex, and concurrent and it is necessary to have mature tools to program them. By concentrating on replicated and well-structured applications researchers fail to advance into the required tool space.

6.4 Computer-Aided-Software-Engineering (CASE) Review

For sequential computing, CASE tool vendors define the phases and processes in the software engineering process in their tools. Figure 6 - 1 gives one vendor's view of standard components. When used to complete each cycle of the Spiral Method, a CASE system is a useful tool for management of the development process. The CASE process shown has a critical assumption: that the hardware and software of the system can be separated by interface constraints. The CASE process assumes that the hardware will be available for execution of the software when the testing phase begins. System design analysis is supposed to establish that the hardware system meets performance requirements. The complexity of interconnect structures makes this a formidable task. Lightly stressed interconnect structures require less demanding analysis. Otherwise, the latency and demand constraints are nonlinear. Physical constraints in C³I systems sometimes limit the designer to reducing the interconnect capability until interconnects are well used. Standard processes and CASE tools do not provide the necessary interaction mechanisms between system hardware and software design to accomplish this task. C³I applications are too complex and dynamic for tools that require one to build large grain modules. In von Neumann architectures, the rapid advances in performance and capacity are powerful compensators for these errors. CASE developed systems often succeed due to the factor-of-safety provided by technology advances. One reason for the spiral method's success is that hardware design and software designs merge along with the user and developer's perceptions of the system.

The insertion of parallel technology negatively affects the validity of the CASE process and its assumptions. The interaction of interconnects within a parallel system is not suitable for closed form analysis for applications of the complexity that require CASE tools. Software

developers must consider the details of the hardware organization and bandwidth capacity along with the software effort. Setting constraints (or goals) for software modules is no longer a suitable method of separating software and hardware. To correct this assumption, CASE systems need to allow a similar process to integrated circuit development tools. The electronic design automation process offers a model for these changes.

Tools for electronic design automation provide a paradigm useful for the software engineering process for parallel systems. Figure 6 - 2 shows that process.

6.4.1 Comparison of CASE and EDA

Electronic-automated-design (EDA) process tools use system interaction models as an integral

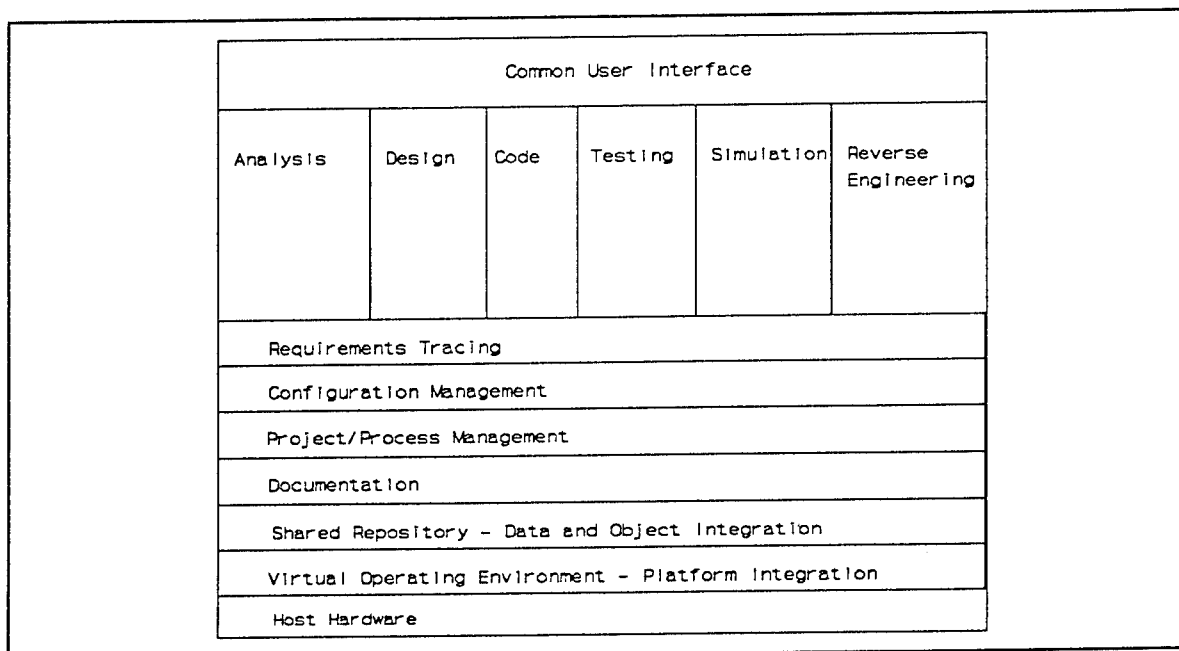


Figure 6 - 1 Computer Aided Software Engineering

part of each design phase. Languages that support the EDA process allow design expression at all levels: behavioral, register-transfer, standard cell, gate, and transistor level design expression. At each stage simulators execute the design and provide functional tests and quantitative timing and resource responses. At each stage, the designer defines the organization, placement and interconnection of the components. Mapping to a specific foundry occurs at the gate-to-transistor interface, allowing portability of the design to various sources. Foundry-rule libraries are the analog of the software developer's use of extensions to standards. Thus gate-to-transistor mapping is based on a standard set of gate constructs that are the EDA user's equivalent of a virtual machine. Engineers use automatic and hand guided tools in the last stage to match foundry design rules. Object oriented methods are commonly used throughout the process by vendors of EDA tools. During each transition between phases, engineers generate a set of tests and test vectors. Those tests provide a decision basis for passing to the next phase. Simulators of the actual operating environment

are often used to place the design under expected stress. Test vectors for a computer chip come from running code of the type expected. For example, RISC computer designers use SPECmarks as the test generator. SPECmarks are performance indicators but are also used as test vector generators. Millions of lines of code run through the simulator to ensure that a processor chip is adequate for foundry insertion. Foundries guarantee that the chip produced will work (function, timing, and power) according to the test vectors used to accept the chip for production. *In both EDA and Spiral/CASE, the process advantage is the early resolution of the imprecision of initial specification.* The EDA system has the advantage that the target instruction set(gates) is significantly more restricted and well defined than that of the CASE system.¹

Separation of hardware and software development is important to the engineering of parallel hardware and parallel software. Without this capability each effort by hardware vendors and software developers is ad hoc and has little potential for large markets necessary for commercial advantage.

6.4.2 Influence on Software Engineering for C³I Systems

Software engineering deals with large, complex, dynamic, mixed hardware systems that have a long useful life. Parallel research technology now exacerbates the software engineering process.

Performance and programmability on large scale, complex, concurrently operating components are issues important to C³I systems. Such systems require multiple organizations, large programming teams and a hierarchy of skills and knowledge for their carrying out. They operate and change over a long period - the life cycle. The systems are heterogeneous and contain a mix of computer types ranging between special developed processors and commonly available commercial workstations. The systems often operate over a wide geographic range. Their components may be airborne or at fixed locations spread across a wide area. The complexity of development requires software engineering methods. These methods define a process for managing the complexity of the system and the information exchange between builders. Software engineering processes provide a rational means of dealing with these systems. For example, the Spiral Model (Boehm) provides one way of dealing with the risks of imprecise user needs and unknown external or environmental factors.

Ever increasing demands for higher performance is a common characteristic of large systems,

¹EDA tool vendors are now in the process of reengineering their tools because the wire delay time can be ignored in feature sizes above 1 micron. Now that gate delays are not the dominant delay the accuracy of predicting timing delays across a circuit becomes less precise. The separation of gate delay from layout delay provided a hidden layer in prior versions. (The layer following, foundry rules, is much like the compiler action on intermediate compiler form.) Their debate is whether the logic designer should learn layout methods and penetrate the layer in their next versions to get more repeatable timing delay predictions. The problem appears to be similar to that of "mapping" to architectures.

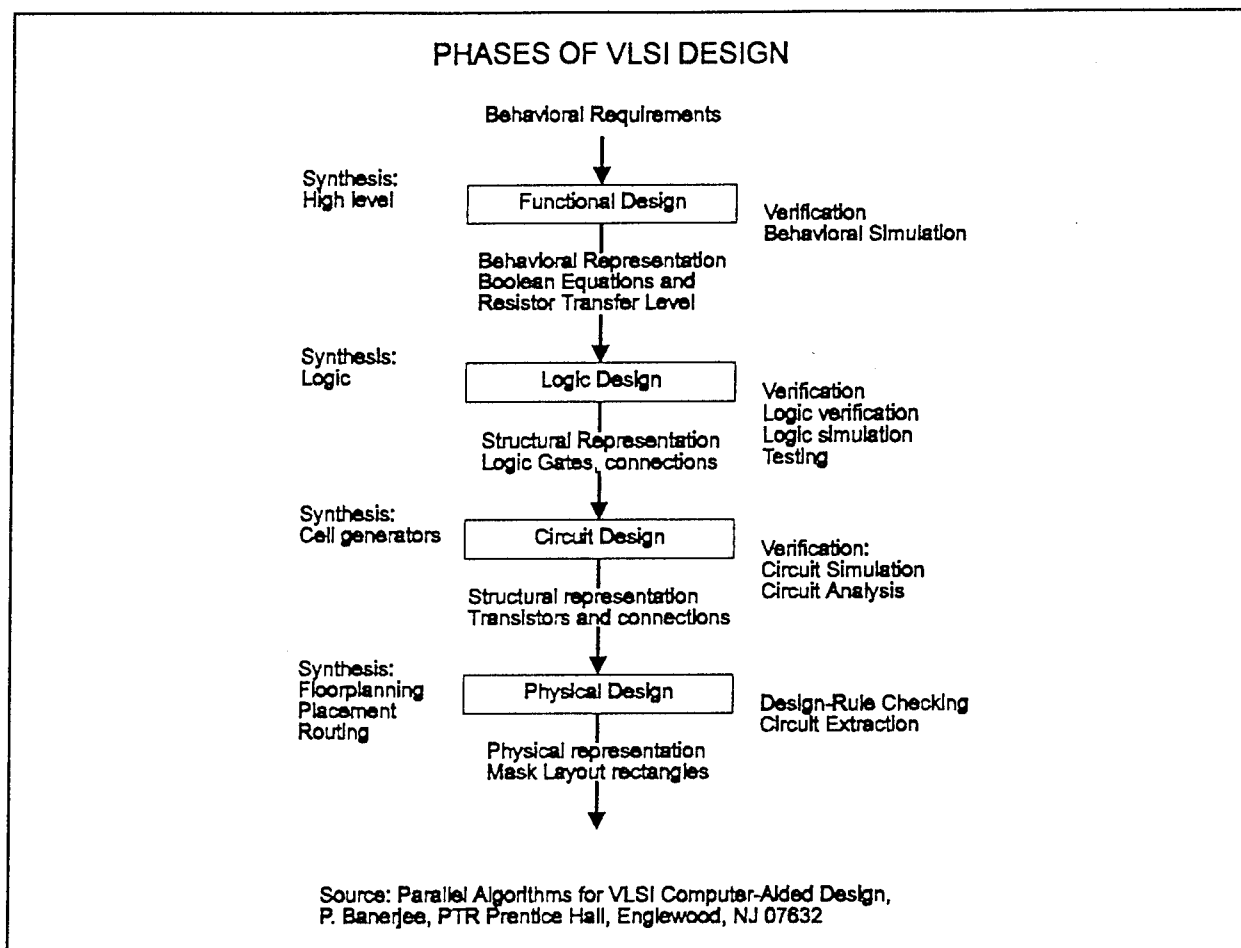


Figure 6 - 2 VLSI CAD Process - Quantitative Results at Each Stage

both military and commercial. New missions and threats often require continued performance expansion. However, they remain constrained by the existing system. The excellent progress in workstation performance and human interfaces and standards (e.g., Common Operating System Environment or COSE) leads to higher expectations from the user. Now she can easily create complex requests and expect the same rapid response as prior simple screen constrained requests. This higher expectation leads to demands for higher performance and new operational actions and responses. The 35% growth in processor speed each year of workstations and microprocessors provides one performance deficit solution. However, many systems now have performance gain assumptions built into the system design. Parallel computers are a solution to the performance demands.

6.4.3 Software Engineering Project Metrics

For conventional computing, software engineering processes have a factor-of-safety due to resource capacity and performance growth provided by the rapid progress of technology. However, parallel systems violate assumptions of conventional software engineering processes. Rapid obsolescence of parallel computers, long procurement cycles, the lack of a quantitative model for performance assessment effects these processes. Integrating quantitative methods into the parallel software development process is necessary for an ideal environment where each phase accurately reflects the expected performance of the system being built. Enhancing the spiral method for performance modeling at each phase of development for parallel computers is one approach. To accomplish this goal there is a need for a programming model that is architecturally independent which couples to quantitative methods of performance evaluation based on instruction timing. Such a programming model either requires bandwidth-scalable interconnects for every target parallel machine or requires development of analysis techniques allowing compilers to deal with the nonlinear response of interconnects.

6.5 Expressing Parallelism

6.5.1 Parallel Machine Models

The discussion that follows is for complex parallel programs which have dynamically changing requirements. Well structured, data partitioned algorithms may not have the same requirements.

It is commonly believed that shared memory and distributed memory systems require locks/monitors and message passing respectively as programming models. This is not necessarily the case. The programming styles of shared memory systems with implied freedom from mapping and distributed memory message passing both have advantages. Shared memory systems provide a common name space but require that shared access be controlled by locks and monitors found by many to be difficult to program correctly. Many reliability problems result from the difficulty in correct and provable codes for shared memory systems. Message passing automatically provides protection and no locks and monitors are required for correctness. Message passing is conceptually less difficult to program. However, message passing is believed to be difficult because it is blamed for the mapping required to achieve good performance in distributed memory parallel computers with very high overhead for creating and delivering a message. If messages were as efficient as a shared memory access then the user related problems of message passing would be eliminated. The system vendor would have to provide a method of naming across all the processors with protected space for each thread of execution. This point of view points out that weak parallel systems capabilities (high latency and low bandwidth) has led to the wrong view of message passing.

Shared memory multiprocessing can become common place and widely used if the software reliability cost of locks and mutual exclusion operations are overcome. The solution for a shared memory system is protected threads of operation created by the compiler and

operating system can allow the protection of the message passing systems. Likewise, with adequate bandwidth and low latency for delivery of small messages the distributed memory systems can perform without mapping. Distributed memory systems with hardware support for interface to the interconnect system can mimic shared memory. Both are viable and a common programming model (or models) can be built on both if adequate hardware, compiler and operating system performance and features are present. The result is message passing protection for the private context between threads of execution with the common name space of shared memory programming.

The ideal programming model is a message passing interface which hides the architecture of the multiprocessor and provides its user the private memory safety of locally protected memory for each process. The hardware and operating system could be shared memory or distributed memory. A criteria for existence in the marketplace will be that the specific architecture would be invisible, in fact might change over a single manufacturers line (e.g. At&T's symmetric multiprocessors up to 32 extended by distributed architecture up to hundreds). This interface should allow small as well as large messages to be created and transmitted. Thus, on shared memory machines small messages of single memory accesses would be better supported directly. On distributed memory systems low latency software protocols and hardware interconnects would allow similar message sizes. On larger machines multithreading (switching in another task) would be used to hide latency of the interconnect. Without hardware and support software that meets this model a common programming model is difficult. (This discussion does not include clusters it applies only to closely coupled processors. A second programming model is required to increase the granularity and hide latency in clusters. However, clusters are best used for multiple copies of an application for throughput, not for running a single application at its lowest turnaround time.)

The future of parallel compute models is that one will emerge in a short time and that systems vendors will then use the individual operations of that model as the test of performance. They will add the extra hardware necessary for some markets and leave it out for others, but the addition and subtraction will be on a quantitative basis. Thus, architecture and programming model will be separated.

Programming models for message passing and directory based shared memory have a dilemma if they must support both high and low latency architectures. High latency systems should be considered a form of clusters. Low latency ones should be candidates for single application rapid response applications. Computer researchers should concentrate on providing a model that operates on low latency systems independent of architecture type. In addition, the programming environments might prevent use of the same tools on both low latency and high latency ones. This barrier makes it clear that applications for high latency systems would not execute effectively on high latency systems.

The software engineering community needs a programming and software engineering process that provides portability across both architecture types. There are two choices:

- ◆ Emulate the shared memory model in capability-scalable computers via directory emulation
- ◆ Build an effective small grain, message passing model for both SMP and capacity-

scalable architecture types

Effective operation of either choice is technically feasible for processors with adequately robust interconnect (high bandwidth and low latency).

6.5.2 Cluster Programming Methods

Message passing is necessary in clusters and farms. The protocol latency in clusters and farms requires application selection or mapping to get a high grain. This mapping process gives message passing a poor and undeserved reputation for difficulty and poor performance results. However, the difficulty in programming is due to the necessity for masking high protocol processing and not to the message passing paradigm itself. Languages for clusters and farms provide a message-passing interface but leave the mapping to the user programmer. Parallel computer vendors now commonly provide Parallel Virtual Machine (PVM) as a message-passing interface for clusters. PVM is likely the most widely used.

PVM is widely ported and now operates on the following:

- ◆ local memory based multiprocessors (e.g., IBM's SP2)
- ◆ shared memory multiprocessors (e.g., Sun Microsystems servers)
- ◆ clusters
- ◆ networks of computers

This does not mean that applications that run well on one type will run well on another. Consequently, PVM ports provide functionality moves between architectures but not performance ones. For example, the mapping necessary to get good performance on a network of computers may reduce the processor count and speedup on the more tightly coupled multiprocessors. Similarly applications running on a large processor count IBM SP2 or Cray Research T3D would take advantage of the low latency. These applications would run poorly on any of the loosely coupled parallel machine types. Applications that do not maintain the level of grain (latency-ratio) expected at the network of computers mode cannot move across architectures and obtain good performance.

Researchers in high level languages for clusters and networks now concentrate on accomplishing general usefulness for arbitrary and multigrain applications without user programmer mapping. So far this is not a reachable goal. Products from Scientific Computing Associate's (Piranha), Parasoftware (Express), and Meiko (Computing Surface) provide programming tools for replicated applications. There are also wide area programming tools, such as, ISIS, TaskBroker and Cronus. Oak Ridge National Laboratory distributes and supports the Parallel Virtual Machine (PVM) tool. NQS (EXEC) is a tool for programming of high throughput applications. A drawback of the cluster is the significant cost of software for replicated applications since each station requires a license.

6.5.3 Heterogeneous System Programming Tools

Tools and mechanisms are also available for clusters of workstations and super large grain heterogeneous computing. These methods, protocols, networks, etc. apply directly to C³I systems. For conservative uses, there is an established software engineering technology for dealing with clusters and large grain-size, heterogeneous systems. Products such as ISIS, Cronus, Linda, PVM, etc. can provide the necessary mechanisms for spreading multiple copies of applications across and managing the resources and processing among clusters.

Distributed cost-scalable multicomputers are more closely connected than the network computers and less so than the capacity-scalable multiprocessor. Their vendors call cost-scalable ones "scalable parallel processors (SPP)." Their rationale is that the addition of a module of processing power costs the same no matter the size of the machine. This essay calls that form of scaling cost-scalable. However, since bandwidth capacity does not grow with processor count, an application that runs on one size machine does not necessarily run well on another. To run effectively on any size the computer must have a property of reduced bandwidth demand per computation as the application size grows. Applications that fit this model are "data parallel" since data partitioning reduces communications significantly. There are several components of applications that have been found that fit this model. To make applications feasible on these machines they must be data partitioned (mapped) to minimize use of bandwidth capacity. Technology influences this approach by allowing the bandwidth capacity to grow in proportion to the computational demand. However, this growth must be faster than the speed growth of the microprocessors. Each of these changes affects the mapping of an application to the processor. Code portability is assured for only pure data-parallel applications. When communications are present, the nodes share the system's bandwidth capacity. Performance often falls off rapidly and differently on different size machines.

6.5.4 The Problem of Application Selection

The tools and parallel computers have become specialized for a set of applications that fail to represent the complex and dynamic executions characteristic of C³I systems. The parallel industry has selected application types and as a result the process of evaluation of both tools and parallel computers has been distorted. Computers must be judged based on a broadly representative application set because the builders cannot anticipate the applications to which they will be put. By concentrating on an applications set that can be mapped to cost-scalable architectures, researchers have created non-representative methods, tools, and evaluation criteria for parallel computers. The result is an inverted rating scheme where software tools are expected to fill the missing capacity of multicomputers that are suited only a structured and stable applications set. Application selection cannot drive the hardware if the hardware is expected to fulfill a wider role than its design criteria.

Some still expect that software solutions can be found to solving problems of inadequate bandwidth capacity growth in cost-scalable multicomputers. As long as mapping is required, extensive applications cannot be ported with dependable performance expectations. General source compatibility of software for general and arbitrary applications across different cost-scalable multicomputers is not feasible unless a common denominator is found. That usually

will compromise the performance potential for all the multicomputers. The structure, latency and bandwidth capacities of multicomputers differ too drastically and are often inadequate for any but their selected application sets. Some cost-scalable multicomputers are simply inadequate for portability of applications, each one must be mapped carefully to obtain reasonable performance. Also, the research community often judges software tools by how well they meet the widely varying demands of a wide range of available machine architectures and sizes. Yet these multicomputers are not suitable for applications beyond their suitable applications. Such an expectation is unrealistic.

Software programming tools for cost-scalable multicomputers are Express, PARMACS, Linda, and PVM. This same tool set is also used on networks-of-computers, which have very high protocol latencies. Applications that perform well on networks-of-computers would be expected to do well on multicomputers due to a reduction in latency. Since only independent and very large grain (infrequent communicating) tasks can be programmed on networks-of-computers these are portable only in the direction from clusters to cost-scalable multicomputers and not in the other direction. However, expectations can vary. A cost-scalable multicomputer is often expected to provide a more dependable execution time since the interconnect structure is in a dedicated backplane. The network-of-computers expectation may well be only to keep idle workstations busy on an application during evening hours. In one case wall clock time is important, someone is waiting for the result, in the other only completion by morning.

6.5.5 Object Oriented Methods

If we assume that applications for industry will bring the parallel computing arena to life the programming methods will be quite different from the academic research that has dominated MPP research computers. Industry uses programmers that tie together shrink wrapped applications from independent software vendors and develop few applications from scratch. When they do have applications and when they are the independent software vendors portability and reusability are important features of the application development. Object oriented software engineering methods and languages are critical to their success. In addition, the applications are likely to be more complex and dynamic than present MPP methods allow. Therefore, parallel technology will have to provide high performance support for applications expressed in C++ or other object oriented languages. Many commercial firms would require even simpler languages that are more easily learned because the individuals that do the applications and know the corporation's information technology are Cobol/ SQL and Fortran based.

6.5.6 Graphical Programming

The requirement for graphical programming is critical to the wide spread acceptance of parallel systems. Graphical system users can be relieved of detailed knowledge of both the parallel architecture and even some of the programming language details. Such users are necessarily high level users that are building application by integrating lower level constructs. Programming using graphical methods should be a template and button bar oriented approach where the parallel language commands and structures are available for integration into a working application by the user. A user would be requested to provide the

logic for cases, loops, by entering it into a window which guides them through any language specific logic expression details. Good programming practices can be enforced automatically by querying the user on the action to be taken for exceptions, etc. The goal of programming should be to provide a very high degree of reusable modules. For a project the engineering of the detailed modules would be left to a small set of power programmers with others simply integrating the detailed objects into the specific application. However, graphical programming must express the "problem architecture." Graphical mapping tools intended to match an application to a particular parallel processor defeat the portability virtue that should be available and are applied at the wrong level, only power users should be doing the detailed mapping required to map the common button bar commands to a particular processor. Those commands should be standardized as a virtual machine programming model and template and button bar calls provided on the graphical screen. Only in this fashion can the benefits of porting only a small set of commands be gained by the users and vendors.

6.5.7 Debugging and Performance Tuning

A separate element is the need to provide debugging and performance tuning. These tools provide instrumentation and constraint probes of the parallel application to control the operation to verify correctness and to eliminate performance bottlenecks. Debugging and performance tools should have a proven interface to selected operating system(s) and an interface to the parallel system's "run time executive" that controls the operation of the application on the system and which provides the implementation of parallel intertask commands. Measures should be provided for the following:

- ◆ individual time delays for each of the parallel commands provided as experienced in the application
- ◆ time delays for the average mixture of parallel commands of the application
- ◆ concurrency - count of the average number of runnable tasks
- ◆ instantaneous concurrency - count of runnable tasks during a step
- ◆ slackness - the concurrency minus the number of processors
- ◆ granularity - ratio of normal instructions to parallel instructions
- ◆ potentiality - count of waiting instructions (total and by value)
- ◆ kernel accesses - count of access to kernel interface
- ◆ external accesses - count of accesses to operating system
- ◆ context switch count - the number of processor swaps (by application, kernel, or operating system)

6.5.8 Performance Tools

Performance tools should provide a post mortem or analysis after completion of application. They should provide all the information necessary for performance prediction - efficiency and speedup equations - based on the measured timings of the parallel instructions and their mixture within the application. The tools should make a recommendation for processor count based on concurrency and provide a processor architecture timing data base.

6.5.9 Correctness Tools

Correctness tools should interface to a commercial debugger that is commonly used in normal workstation programming. There should be history files created during debugging and constraints on execution that allow repeatable (deterministic) operations when active. This control requires control of program re-execution between set break points, or between barriers, or between identified steps in the execution. A history of communication events is a necessary element of the data base that must be provided to control these operations.

6.5.10 Visual Tools

Parallel programmers must manage enormous quantities of information and complex program behavior. To understand these operations visual aids are one solution. To be effective they must provide a proven interface from graphics programming to object oriented languages. Communications must be displayed in such a manner that correctness can be evaluated. Complex interactions should be capable of being built from primitive proven ones. Execution graphs should be displayed between pairs of barriers created to investigate errors or performance bottlenecks. Tools should allow easy and structured building of a concurrent application from a small set of objects defined to perform the application.

6.5.11 Power Languages

Power languages are needed to provide expression of complex operations that cannot be defined in standard languages with expressiveness for simple data partitioning and process interaction structures. Such languages must provide execution of the power expression and provide a performance prediction of the expression on a chosen architecture. A good example, is Proteus, from the University of North Carolina.

6.5.12 Intelligent Interaction

Advances in the way computers interact with humans are providing new opportunities for building systems that are dispersed in unique and different locations. Wireless terminals, power pens, voice interaction, point-of-operation terminals, etc. may provide significant opportunities for systems building that were not previously available. The interaction of humans with a parallel execution should be a goal since human guidance could lead to a more effective solution in both quality and time of execution. Interactive control requires an extreme of dynamic resource and computational scheduling. It also requires that input/output in multimedia forms are required to be integral parts of the computation.

Parallel Software Engineering Assessment

7 Issues in Parallel Software Engineering

7.1 Parallel Software Engineering Issues

Lack of tools and software has been the battle cry of today's parallel system builders when expectations are not met. This is true, the software engineer needs a set of tools that demonstrate the need for more generality and capability by parallel systems for dynamic and complex applications. Without the right tool kit the software will always be blamed.

7.1.1 Process and Design Metrics

One of the major weaknesses in parallel software engineering is the lack of a staged process that can assure the system designer that the performance expectations of parallel computer based components will meet their performance and program goals. Ideally, the tools to support performance delivery and program goals would be based solely on input of a set of characteristics of an arbitrary application set that provides the C³I system required functions.

7.1.1.1 Quantitative Performance Assurance

Performance is measured against the requirements of the C3I system. System designers take this to mean more than raw speed. They include features such as availability, physical constraints, real time response types as parameters in performance requirements. Software engineers should have the capability to define the characteristics of an application, possibly based on an existing version implemented in a different computer technology, possibly based on the observed characteristics of similar systems. After defining those characteristics it should be possible to use a set of standard parallel instructions to predict the performance of different parallel architectures. The characteristics step requires a model of computation that quantifies the characteristics. Possibly those given in the application's characterization essay. The instruction set requires a virtual model of parallel computers and instruction set that is complete and sufficient to model both shared memory and capability-scalable buffer transfers or message passing. The performance prediction requires a level of robustness in the computers to avoid non-linear bandwidth effects. To complete the system feedback input is needed from several different system implementations. A software engineer needs a measurement scheme for testing and feedback of how well architectures deliver operational speed according to prediction.

7.1.1.2 Quantitative Program Assurance

A development program goals are measurements of skills, schedule and cost budgets. This tool gives engineers a metric for difficulty of development based on the application and on a selected architecture. This tool would provide the ability to make choices about the allocation of costs to hardware or software development. The staged development process needs a process similar to the VLSI process where each synthesis stage is verified and tested for functionality and timing before the next stage is synthesized. A component based system

design would have the execution characteristics of each component defined and matched to the chosen parallel processor. A simulation of the characteristics and the architectural instruction timings could be used as verification. At the next stage the component design synthesis would be measured according to characteristics of the module design using modern CASE simulation tools. At the next stage a detailed module program using only the generic machine instruction set would be used to write the module. Simulation and analysis would again verify performance goals. The last stage is the one in which the target machine knowledge is used to increase performance. (This matches the VLSI design rule stage where foundry information is first imposed.) This approach would be criticized by parallel advocates of many of today's machines because the effort of the last stage is significant. The value in the approach is that machines whose instruction set closely matched the output of the third stage would be more easily programmed. Those machines could be expected to cost more since additional hardware capability is required to allow them to perform that instruction set well. However, the experience to date has shown that the investment in operating system software and the attraction of independent software vendors favor the economics of parallel machines that provide availability and robust interconnect structure capability.

7.1.1.3 Development and Life Cycle

An ideal measurement of a parallel computer is to evaluate its impact on the system's development and life cycle costs. Parallel computers potentially eliminate many other computers in the system. Their high execution rates can make a system feasible as well as cost effective. However, if the cost of software development increases to the degree that the hardware savings are insignificant then the parallel computer is hardly cost effective. Comparisons of parallel computers by evaluation of mission life cycle costs could be telling factors in an evaluation. In general, those systems that are the most quantitatively predictable and programmable will lead to lower support and minimum effects from assumed mission changes. Evaluators set criteria from mission need statements. They need a process for relating mission need and life cycle costs for alternative parallel solutions.

7.1.2 Application Metrics

In order to make any quantitative approach to parallel software engineering the application must be characterized and related to known performance impacts at each stage of development. Measurement tools for all aspects of characterization would have to allow estimates of characteristics, models of characteristics, benchmarks measurements of characteristics, and actual code characteristics as inputs. The effort is equivalent to rebuilding the electronic automated design tools set, the associated libraries of behavioral models, benchmarks, and timing estimation tools.

7.1.2.1 Physical Constraints

Additional factors arise in C³I systems that complicate the analysis of the potential use of parallel computers. These include limitations on size, weight, volume, and power consumption. These constraints make it difficult to analyze the application, the system and the computation expansion characteristics of the application. The typical use of many parallel computers as compute engines for single large applications lends itself to being matched to

C³I system requirements. Embedded parallel processors are also traditional for some portions of a C³I application. For example, processing data at its collection point to reduce very high data collection rates down to reasonable internal system communication rates.

Tools would be required that measure the resource consumption of various choices as well as their performance potential, programmability and program suitability.

7.1.3 Performance Delivery & Metrics

7.1.3.1 Evaluation Criteria

The purpose of parallel computers is usually to provide higher speed. Speed can mean a higher throughput for independent transactions or it can measure the rate at which a large single application can be completed. Speed can mean response time to an external interrupt during execution of the single large application. Engineers use other measures of speed that match specific C³I system needs.

This does not mean that engineers only use speed as the only performance criteria to judge a system. Consequently benchmarks include measures other than speed measures. The suitability of a parallel computer for a particular system depends upon many criteria. Commercial organizations insist upon proven availability and portability before even considering speed performance factors. The following are evaluation characteristics for parallel computers:

- ◆ Availability includes stability, reliability and fault tolerance of both the hardware and operating system
- ◆ Programmability includes portability, reusability, relative development ease, consistency with normal programming methods, and performance prediction based on software measurements
- ◆ Physical Constraints includes power, weight, volume, sizes, electromagnetic noise, etc. as deemed necessary for meeting mission goals
- ◆ Life Cycle Costs include hardware, application development, long term support over the entire life cycle
- ◆ Delivered Performance includes the following measures: nominal delivered throughput rate for independent transactions, computation-rate on a single application, responsiveness to dynamic overloads, and real time response
- ◆ Quantitative Performance and Risk Assessment - include accuracy and sensitivity of quantitative measures that portray delivered performance, schedule and cost estimates . .
- ◆ Application Access - includes common applications, tools, languages, operating systems, standard open system interfaces

The users of computers want computers that solve their applications reliably and swiftly. Users do not care about the type of computer. They are concerned only that the system meet the requirements criteria derived from their mission statement. Benchmarks for parallel machines must include measures of the integrity and actual delivered performance of the machine on similar applications.

and actual delivered performance of the machine on similar applications.

Availability

Availability is a test that requires long test periods while running complex applications to stress all the features of a computer. Availability includes operation of all system software and hardware. C³I tests of availability are based upon mission criteria.

In commercial application's users expect parallel processors to match the positive experiences of the user with workstations, servers, supercomputers or mainframes. They expect hardware to be reliable and fault-tolerant. Their high availability expectations require that parallel systems allow replacement of failed modules without full operational loss. Similarly, they expect operating system and application software to be reliable and repeatable. C³I systems have more extensive requirements for fault tolerance and recovery since human life and mission success may depend upon the availability of the processor.

The present status is that most "massively" parallel machines meet research environment expectations, not commercial ones. The availability and integrity levels experienced by early users do not meet the needs of either commercial or C³I demands. Some machines from the business community do meet commercial reliability and fault tolerance acceptability. For example, AT&T's NCR 3600 has dual paths in a folded network with on-line module replacement to give reliability. Data base applications such as Oracle and Sybase are available on the NCR 3600.

Delivered Performance

General performance delivered by a parallel machine depends upon several detail level factors. The way those factors change with changes in the number of processors, $p(n)$ is a primary consideration. The factors and the primary influencing design issues are:

- ◆ The effective execution rate of the node computer on sequential code operating as a standalone processor, cache memory, local memory, access to input/output at the node
- ◆ The effective node execution rate within the parallel system with no application intertask communications, i.e., the processor effectiveness during interconnect latency (for both multitasking and multithreading), the link bandwidth from the processor to the system, uniformity of access to input/output and operating system overheads
- ◆ Performance of the interconnect structure - capacity-scaling, small latencies (application intertask communications timing)

Appendix gives additional detail on stand-alone node, node-in-the-parallel system, and interconnect structure performance.

7.1.4 Programmability & Portability

The lack of a common consensus on an approach for decomposing, mapping and synchronization and on a common programming model (shared memory, message passing, explicit use of virtual shared memory) leads to a situation that anecdotal results are used to defend one position over another. Each case is ad hoc. The work and effort of parallel programmers is not appreciated because it suits special cases, not a general principle on which others can base their decisions.

7.1.4.1 Programmability

The von Neumann model gets acceptable performance for applications programmed with functional goals and no great concern for performance. This initial performance can often be improved using a mix of nonportable methods. Some programmers take the extra effort and accept the long term maintenance disadvantages of nonportability. They typically get a performance improvement of a small constant factor. Parallel computer programming should be no different. Programmers should obtain acceptable performance with good portability with a moderate first level of effort. They should expect that additional performance obtained through mapping would be nonportable.

Portability is one measure of programmability. The industry problem of attracting the independent vendor to MPP computing is the symptom that programmability is far below acceptable standards. Present programming models (PVM, Linda, etc.) attempt to serve for both networks of computers and for tightly coupled computers. Such a broad attempt yields a program that cannot provide acceptable performance on either target machine class across a wide range of applications. An application design, which serves widely different machine types, compromises the programmer and requires extra effort to map to very large latencies for the network machine. This mapping reduces the concurrency that could have been expressed and executed in the tightly-coupled, low-latency machine. This reduced concurrency limits the performance potential on the low latency machine. Therefore, dealing with a wide range of latency for different parallel computer types is not a realistic goal. Advocates of portability techniques do not clearly define the domain of portability for their products. They should distinguish their product's suitability for both networks of computers and types of parallel computers.

Leaving networks of computers aside, we find that some of the available parallel machines do not deliver acceptable performance except on carefully mapped applications. Efforts to create a portable software structure should not be considered a failure if the parallel computers are capable of effective operation on only replicated copies. Portability measures of architectures should distinguish these architectures and reduce their evaluation rating. Such computers are not much more than a network of computers on a backplane.

Automatic compiler success is another measure of programmability of parallel computer architectures. An automatic parallel program generator that gives effective performance would be an important advance. However, recent experience shows that efficiencies are too

low on many parallel machines for automatic program analysis. Present data decomposition, necessary to use high performance compilers, has lead to disappointing results unless the applications were limited to highly structured ones.

Evaluators of parallel software systems say that porting effort is inversely proportional to the latency of a parallel processor. Decreasing latency impact by scaling the application is often a solution for only part of the application. Engineers make careful designs of interconnection systems. They seek to reduce the time spent for message transfer into and passage though the interconnect. Their efforts are in vain if delays for communications protocols dominate the total delay. High latency causes the programmer to build applications with higher computation content than communications. This effort reduces the opportunity for parallelism and limits the application's potential for higher speed through parallel computing. The use of coprocessors to overlap the protocol latency with computational use of microprocessor has not been as successful as one would expect. This is most likely a reflection of a concentration on large-message passing as a model instead of packet size message passing.

7.1.5 Relation of Needs to Capabilities

The most important issue in parallel computing is that mission critical application capability is missing in both government and commercial uses of parallel computing.

Like military C³I systems projects, the large scale mission critical applications built for commercial decision support applications have required excessive labor and maintenance and risk of performance. A software engineering goal is to find a solution for this daunting problem. Mission needs for quick results to a complex decision that solves battle (or market competitive) pressures is the area that the most gain can be derived. The commanders that make these decisions are willing to pay for solutions in their decision making process. They care not for the technology used, but are unwilling to change their needs to match the computer's shortcomings. The present approach of selecting applications due to their suitability to an architecture has limited value added. A significant payoff would result from use of robust capability-scalable parallel computers. Interfaces to legacy systems are necessary so a balance of I/O and computational capability is needed. Software engineering practices are feasible for computers that deliver promised performance on portable programs with high complexity and dynamic action. Software engineering practices cannot solve their applications solutions if the computers lack the capability to provide adequate support for mission critical applications.

7.1.5.1 Suitability Uncertainty

Another major issue in use of parallel computing is the uncertainty of application suitability and parallel performance benefits in the face of the complicated architectures and unknown application characteristics. Deciding when an application is suited to parallel computing is difficult. The characterization of many C³I components in the application assessment essay concluded that the components have high parallel performance potential but not on many of

the available machines. Those machine have resulted from a different set of goals than that of C³I and most system builders and of most users in industry.

7.1.5.2 *Conflicting Measures*

An application with fine synchronization grain and high parallelism uniformity is one for which parallel instruction communication delays become insignificant. (Terms are defined in the characterization method in the application essay.) However, the process of creating applications with large grain is difficult. It requires selection of algorithms, mapping and often unresolvable decisions about dynamic portions of the application. Most often the degree of parallelism is reduced as the synchronization grain is increased. The uniformity of parallelism is aided by having many small grain tasks.

Note the conflict between the efficiency attained by large grain creation and scheduling improvement gained by high and uniform parallelism. With more tasks the parallelism is high. When programmers combine tasks to create large grain modules, they reduce the number of tasks available for easing scheduling inefficiency. The solution: increase the computation-to-communication ratio without decreasing the task number or affecting their size differences. The result is that many parallel processors are capable for selected applications where each task is identical at each node. Data is mapped to the nodes to keep communications low. This type of processing is called single-program, multiple-data (SPMD) and is typically the mode in which a cost-scalable multicomputer is used).

7.1.6 Breadth and Market for Applications & Tools

A key measure of a parallel system is the number and diversity of independent software vendor tools and applications. Defining a set of application types that match the needs of C³I subsystems is a useful exercise. The set is a qualifier for a parallel computer. The performance of those application types on the processor provides a measure of how the parallel computer might do on a future C³I system. Such matching would require a wide range of application types to include the diverse needs of C³I systems. The selection could include process control, transaction processing, data base, engineering simulation, economic modeling, etc. An important tool would be one that measures the concurrency, grain, complexity and parameters of these applications to allow comparison to the content of C³I systems.

Parallel computers have been accepted enthusiastically by only one application vendor - the data base manager server application. The most common data base management systems have been ported to a number of parallel computer of all types. This is due to the use of these machines as servers, performing the single parallel data base application for many clients on a network. Other applications remain on the workstation being fed their data from the servers.

However, acceptance among other independent software vendors has been weak. ISVs have a hard decision to make on the target parallel machines. The diverse number of architecture requires that a standard model be used for all before fine tuning to machine dependencies. Threads (shared memory programming) and message passing are the two approaches most widely used.

Threads in symmetric multiprocessors are being standardized to POSIX, but each system vendor's use of proprietary methods and structures requires a different design. The competitive forces are so great in the server market place that no vendor is guaranteed to maintain market share. Unless carefully constrained shared memory programming is not portable to message passing machines. Therefore, threads are unlikely to be ported to operate on distributed memory architectures, requiring a second port (and a very different design) for the large processor count machines. Lacking a common model ISVs tend to only port to the vendors that can pay. In addition, many industries are not willing to pay for a second copy of an application in their computer center. The vendor may have to choose to support the vector processor or the parallel processor at a given site. These constraints and difficulties limits access to parallel applications.

PVM and other message passing tools reduce the performance potential of the application because of the high latency costs require building large messages to hide protocol latency. Linda and other explicit virtual shared memory models are in competition with the government's free distribution of PVM. Therefore, they lack the business level necessary to select and finely tune their model on suitable architectures. the combined difficulty of suitability uncertainty, non-portable models, and small market keep ISVs out of the market.

Parallel Software Engineering Assessment

References

- Parallel Algorithms for VLSI Computer-Aided Design, P. Banerjee, PTR Prentice Hall, Englewood NJ 07632
- Computer Design, Special Report: CASE Tools for embedded and realtime applications, Tom Williams, Editor, April 1994 pg. 65-78.
- Henry J. Bush, Presentation, Thrust 3 Program Overview, 21 June 1994
- RADC-TR-89-337, Final Technical Report, January 1990, Rome Air Development Center, Griffiss AFB, NY, Software Techniques for non-Von Neumann Architectures, C. Lightfoot, et al., Computer Sciences Corporation
- Electronic Engineering Times, August 1, 1994, Executive Viewpoint: The process is changing, P. Yeatman, Radstone Technology, Montvale, NJ.
- IEEE Spectrum, May 1993, Focus Report: Workstations and PCs, M. Furtney and G. Taylor, Of workstations and supercomputers, pp 64-68.
- F. Baskett and J. Hennessy, Microprocessors: From Desktops to Supercomputers, SCIENCE, vol 261, 13 August 1993, pp 864-871.
- Computational Applications in Manufacturing, A. Formica, RCI Management White Paper, 1994.
- Clustered Workstations: The Dominant Parallel Architecture?, J. Mohr, RCI Management White Paper, 1994.
- Fundamentals and Practicalities of MPP, G. Astfalk, The Leading Edge, Part 1, August (p 839), Part 2, September (p 907), and Part 3 October (p. 992) 1993.
- A Spiral Model of Software Development and Enhancement, B. Bohem, Computer May, 1988.
- HPC Research Note, October 27, 1993 Gartner Group, H. Richmond, MPP Systems: A Strategic Weapon for Industry
- High Performance Computing: The Potential Contribution to General Purpose Computing in the 1990s., Gartner Group, H. Richmond, 1993.
- H. Richmond, "Commercial Applications of Parallel Processors," Austin, TX, July 1993.
- "Software Engineering of Parallel Systems," Carl Murphy, Encyclopedia of Software Engineering, JWiley&Sons, 1994.
- "Future Directions in Supercomputing," D. Robb, Supercomputing '93.

Parallel Software Engineering Assessment

A. Appendix A Some successful large scale scientific computing results.

TABLE A. High Performance MPP Applications (TMC CM-5)			
Application	Problem Domain	Size	GFLOPS on 1024 Nodes
EFES	Finite Element 3D Flow	1 M elements 1 M nodes	37
TeraFrac	Ductile Fractures	1.2 M elements	27
BGK	Gas Dynamics	Not published	59
SPaSM	Molecular Dynamics	180 M atoms	50
ENSA	Aerodynamics	30 M grid points	18.7
QCD	Quantum Chronodynamics	64 ⁴ Lattice	44

Other MPP systems give promising results. For example, the Intel Paragon reached 102 GFLOPS on an LU factorization code. The Cray T3D and IBM SP2 also provide significant performance on parallel benchmarks. Table 2 compares the result for NAS benchmarks for a 64-processor case.

Parallel Software Engineering Assessment

B. Appendix B: Capability Evaluation of Parallel Computers

Individual Node Evaluation

The designer of a parallel program needs a model of computation and memory access that is simply an extension of the von Neumann model. In the von Neumann model each layer of memory access incurs an additional delay in the instruction used. The hierarchy is: Register to Register, to Internal Cache, to local Secondary Cache, to local memory, to distant memory within the parallel computer, to virtual memory, to local disk, to distant disk system. Software support tools (compiler and operating system) should deal with these delays automatically. Operating systems on sequential computers provide multitasking and multithreading to deal with the delay in disk access. Virtual memory, used to extend the physical memory size, allows the programmer to avoid considering mapping of the problem to physical memory. Users should expect the same of a parallel computer, that is, virtual memory automatically provided by the multiprocessor hardware and parallel operating system.

A microprocessor-based workstation delivers a useful fraction of its peak performance. In this market, performance statements often come from the chip's peak performance while operating at peak speed on register-to-register code. Evaluators quote a benchmark that represents a mix of workstation applications (SPECmarks). This benchmark requires a complete set of supporting cache memory, normal memory, disk accesses, etc. components. The results are not perfect. Vendors publish numbers affected by compiler tricks and use of maximum speed and capacity caches, memory, and disks. However, the results do establish bands of performance. For example, they clearly separate 80486 personal computers from RISC based workstations.

Hardware designers, therefore, have a target applications set upon which to concentrate their efforts. The applications set is too complex to design for it directly. This requires quantitative methods to assess performance of an instruction set and chip design on the benchmark. IBM pioneered the technique. The authors, Hennessey and Patterson, describe quantitative methods in "Quantitative Design of Computers." These methods depend upon establishing a weighing of the use of each instruction, the timing of the instruction, and the typical ordering of instructions. The idea of the Reduced Instruction Set Computer (RISC) computer and their designs are the result of the methods. In RISC computers the most used instructions are primitives of the instruction set. Compilers build infrequently used instructions from the primitive set. The compiler also improves the ordering found in the application mix for the pipeline and register set.

No user of a RISC-based machine expects to obtain the peak instruction issue and execution rate of the microprocessor on real applications. The published SPECmarks provide a better clue to a user's performance. A finely tuned performance estimate would use numbers from a subset of SPECmark s that match one's application. Yet parallel system discussions frequently evaluate parallel computer processing nodes in terms of multiples of the peak instruction issue rates. These discussions should use an effective rate for individual nodes as configured within the parallel system. The Cray T3D, for example, has no secondary cache for the DEC Alpha chip used as a node processor. In addition, its

node memory does not allow the maximum size available in workstations built by DEC. Access to input/output or virtual memory page faults may require delays not found in the standard workstation even when operating on sequential codes. Therefore, there is no Alpha workstation equivalent to a node of the Cray T3D. Some compare parallel computers based on multiples of the instruction issue rate. Such comparisons are invalid but are often used to build unrealistic performance expectations. The single node performance is difficult to obtain because there is no equivalent standalone hardware. The performance measured "in silica," that is within the system is the right number. Running a sequential application on one processor at a time while varying the node location provides an estimate of single node performance in the system. Use of that measure is far superior to use of outside the system node estimates.

In-System Node Performance

This is a measure of parallel computer node performance with sequential applications (no interprocess communications) running at all nodes. It measures the node performance. It is a better measure of the peak speed of a parallel computer. One can establish a machine scaling baseline by measuring the sequential performance for the smallest module through maximum node count available. The baseline also reveals any compromises in input/output at higher node counts. Shared-nothing benchmarks are good candidates for deciding node performance. On-line -transaction-processing benchmarks use parallel computers in the "shared nothing" way. Therefore, they may be ideal candidates for evaluating node performance capability across a range of machine sizes. Thus, one can start with an accurate value for the effective, in system, node performance. Its performance degradation, due to parallel communications, measures the effectiveness of the interconnect system.

Interconnect Performance

Opinions differ about which machine factors are important. One factor that does not appear on the list above is scalability. Vendors misuse this term and use different definitions. Many vendors consider modularity as the definition of scalable. This essay gives two contrasting definitions below.

Massively parallel advocates define scalability as the capability of an architecture to maintain constant interconnect cost (bandwidth capacity) with respect to the number of processors. This essay calls that definition cost-scalable. Others claim that a scalable architecture has constant bandwidth capacity between any two arbitrary processors for any processor count. This essay calls that definition capacity-scalable. Both definitions fail at true scaling. That is, constant execution time of an arbitrary application as it grows in size to match the number of processors. Latency increases as machines grow larger, even if only by a $\log_2 p(n)$. Capacity-scalable architectures can approach true scaling by multithreading to hide latency variations.

Why is capacity-scaling an important feature? If the programmer must consider the bandwidth of the application then the non-linearity of interconnect can cause unexpected and significant swings in performance. Benchmarks can show architectural capability for capacity-scalable operation. Machines that do not capacity-scale can only be suitable for applications of special types. When machines are not capacity-scalable they require selection of special algorithms. Applications must be "mapped." The resulting software is not portable. Machines can have such a large latency in communications start up protocol processing that the bandwidth is not the leading issue. Present massively parallel systems have a start up delay when creating a message to another processor that is large enough to hide the

impacts of the bandwidth saturation non-linearity. However, careful designers would test this assumption before analyzing performance using linear network delays.

Given an adequate interconnect bandwidth, latency differentiates between parallel computers. Latency is the delay incurred when information passes between two processors when the machine's interconnect is very lightly loaded. The validity of using a linear latency method in evaluating the communication delays within a parallel system depends upon adequate bandwidth capacity.

When it is valid to use linear techniques, a useful factor is the ratio of communication time to average instruction period. Communication time is the time to move a single datum from one computing resource to another. That ratio is the latency-ratio for the parallel computer [Astfalk]. It sets limits on the potential efficiency of the application on a processor.

Latency

Latency has three terms:

P - the fixed start up or protocol processing time

W - the time to transfer a 64-bit word out of the memory at one node into the interconnect then out of the interconnect to the memory at the other node multiplied by the number of words

H - the interconnect stage-hop-delay-time multiplied by the number of stages necessary to get the message to the memory in the other node

For typical scientific applications W is found to range from one to 10. In good designs H varies with the log₂ of the number of processors. In good designs the per word transfer time and the hop delay times can be kept below ten instruction times. The start up or protocol time is a latency bottleneck in many parallel computers. This delay can take about one thousand instruction times to do. For example, Intel's Paragon takes about 800 instructions for protocol processing. Programmers hide the costs of protocol delay by making W large. Presently parallel computers require the user to data partition or map the application to create large message sizes (larger W) to overcome the protocol component of latency.

The latency of starting a communication has remained embarrassingly constant when measured in terms of the number of processor clocks taken. As clock periods are reduced inside the microprocessor the protocol and interconnect structure interface operations must be improved significantly just to keep up. For networks-of-computers the latency of TCP/IP is so large that significant improvements are feasible, but for multicomputers and multiprocessors making improvements is a difficult task.

Parallel Software Engineering Assessment

C. Appendix C: Quantitative Performance and Risk Assessment

The increase W solutions presented here apply only to selected applications for which the problem's computational size increases faster than the required communications. Some applications fit the criteria (e.g., fluid dynamics) and others do not (e.g., structures). Therefore, combinations of algorithms with both application types are performance limited. Amdahl's law expressed this limit. When applications are selected to contain only the type that fit they can be scaled. The two extremes provide a least and most optimistic bound on expected performance.

Amdahl's Law - Least Optimistic Case

The speedup promised by the parallel system can be compromised due to failure to do well on even very small portions of the system. Therefore, it is more important that the parallel analysis include all components. The speedup impact is Amdahl's Law. [Amdahl] Equation 1 shows this law for one sequential and one parallel subsystem. To use it one must expand it to include each separate subsystem operation in the system.

$$Speedup_{Amdahl} = \frac{1}{(1 - fraction_1) + \frac{fraction_1}{Speedup_1}} \quad \text{Equation (1)}$$

Where the $Speedup_{Amdahl}$ is the speed gain due to the enhanced function par, $speedup_1$ is the ratio of the parallel component speed to the sequential component speed, and $fraction_1$ is the fraction of the application sped up. The only way to avoid this speedup limit for a chosen parallel enhancement is to increase the concurrency by increasing the term $fraction_1$. Some applications expand their computational fraction as increase in size and attain a higher $fraction_1$. That allows speedup close to one. An excess of task count over processor count (slackness) makes $fraction_1$ nearer to one.

A more detailed analysis of extensive approaches is feasible when bandwidth capacity is adequate. In that case the parallel communication instructions have constant latency values. The factor η , as defined in Equation 2, gives a maximum performance boundary. Performance and efficiency prediction are the critical decision parameters in the analysis level. Equation 2 gives the maximum efficiency in terms of the application's grain, G, the average parallel instruction set time, T_p , and the average node instruction set time, T_o .

$$\eta = \frac{1}{1 + \frac{1}{G} * \frac{T_p}{T_o}} \quad \text{Equation (2)}$$

Scaled Speedup - Most Optimistic Case

Since problem expansion increases the parallel portion of an application it is possible to overcome the limit imposed by Amdahl's Law. Assuming that the sequential portion stays constant as the parallel portion is expanded, the speedup for p processors is:

$$Speedup_{Scaled} = (1 - fraction_1) + p * fraction_1 \quad \text{Equation (3)}$$

This formulation gives a perfectly linear speedup as processors and the application both increase in size. Realistically the actual results can fall between the two formulations. Some portions of an application speed up through expansion while others do not. The ratio of scalable to non-scalable content in a benchmark is a predictor of performance potential over a range of problem sizes.

Parallel Software Engineering Assessment

D. Appendix D: Connectivity of some parallel architectures

The information in Table [D-1] shows the connectivity of some parallel architectures. The purpose of the table is not to provide information that should be necessary for the selection of a computer. In fact, such information should be hidden from the software engineer. The essay on performance evaluation and appendix 5F provides the mechanisms that should be used to determine the effectiveness of a parallel computer for a given application. However, the trends are indicated by the approximate year of introduction.

TABLE D - 1 Connectivity of Selected Architectures				
Company	Model & Trend Date	Organization	Connectivity/ Degree	Comment
nCUBE	(1992)	Hypercube	12	
Intel	Paragon (1992)	2-D Column based Mesh	4	Eight processors/ column
Kendall Square Research	KSR-2 (1993)	Multi-ring with directory based cache coherence	2 + Off - Ring	Off-Ring interconnects have higher bandwidth
Cray Research	T3D (1993)	3-D mesh	6	
Thinking Machines	CM5 (1992)	Fat Tree with reduced bandwidth outside local connections	4 + Off - Tree	Separate SIMD Control
IBM	SP2 (1994)	Cross-Bar	8 paths	
AT&T	3600 (1993)	Folded Banyan	8 paths	Folding provides fault tolerance
Cray Research	SuperServer (1994)	four busses and segmented memory	4 x Busses with cache coherence	Multiple busses provide extension to 64 processors
Compaq, Encore, AT&T, Sun, and PC network servers	(1991 - 1994)	bus with cache control	2	Single High Performance Busses

Parallel Software Engineering Assessment

E. Appendix E: Clusters

Cluster and Farm Networks

The networks used in farms are typically based on Ethernet protocol and physical communication channels. Clusters use high performance Ethernet or Fiber Distributed Data Interface (FDDI) for communications between processors. The latest technology examples are the IBM Fiber Optic Link (220 mbps), the Network Systems Corporation DX series and the DEC FDDI Gigaswitch. HP and Convex also have prepackaged cluster systems.

Cluster Protocols

Protocols necessary to use these networks contribute to the latency of communications. The underlying delay and bandwidth capacity of the physical network often has a smaller contribution to latency than the protocol processing. Because of protocol delay, programmers must either create or select applications with a very large ratio of computation to communications. That type of application is a large grain one. Because of large grain selection or programming, the typical communication requirements are for transmission of large files.

The need for the robust services of TCP/IP in dedicated clusters is not necessary. TCP/IP processing is often processor compute intensive. In some systems, builders use the protocol (IPI-3) that requires less processing and sustains a higher bandwidth. Fifty to Eighty Mbps transfers between workstations are feasible over HiPPI (High Performance Parallel Interface) and Fiber Channel Standard (FCS) paths. A switch bandwidth of one Gbps is now available. These changes put the burden on the physical capability of networks (contention delay and bandwidth capacity) instead of hiding it in processing protocols.

Cluster Cost Elements

Clusters have powerful economic advantages because they use standard, commodity volume hardware and software components. Clusters also have a system software development cost advantage over multiprocessors. No system extensions other than racking the components and installing operating systems and PVM on each processor is necessary. The burden of tools and programming falls on the user. Vendors like to this design because it has high margins and low risk.

Rome Laboratory
Customer Satisfaction Survey

RL-TR-_____

Please complete this survey, and mail to RL/IMPS,
26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and
feedback regarding this technical report will allow Rome Laboratory
to have a vehicle to continuously improve our methods of research,
publication, and customer satisfaction. Your assistance is greatly
appreciated.

Thank You

Organization Name: _____(Optional)

Organization POC: _____(Optional)

Address: _____

1. On a scale of 1 to 5 how would you rate the technology
developed under this research?

5-Extremely Useful 1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please
suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No_____

If yes, please identify the area(s), and comment on what
aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

***MISSION
OF
ROME LABORATORY***

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

- a. Conducts vigorous research, development and test programs in all applicable technologies;
- b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;
- c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;
- d. Promotes transfer of technology to the private sector;
- e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.